MASTER 64

PROFESSIONAL DEVELOPMENT SOFTWARE

Abacus Software

P.O. BOX 7211 GRAND RAPIDS, MICH. 49510

COPYRIGHT NOTICE

WARRANTY

ABACUS Software makes no warranties, expressed or implied as to the fitness of this software product for any particular purpose. In no event will ABACUS Software be liable for consequential damages. ABACUS Software will replace any copy of the software which is unreadable if returned within 30 days of purchase. Thereafter, there will be a nominal charge for replacement.

Third English Printing, November 1984
Printed in U.S.A.

ISBN# 0-916439-21-6

This manual has been designed to be both a teaching aid and a reference manual. Careful reading of this manual is recommended for a better understanding and use of this BASIC tool. The manual has been designed for a quick understanding of the product and also for fast look up.

The chapters with an A.P. (Advanced Programming) label are harder to assimilate and a good knowledge of COMMODORE systems and of MASTER's easier sections are required.

There are 6 main sections MASTER SCREEN, MASTER PRINT, MASTER FILE, MASTER BASIC, EDEX and BASIC 4.0.

The appendices at the end of the manual have been designed to allow quick search with a first level of information and references to the main sections for a second more detailed level of information

CONVENTIONS

1) - Square brackets [ ] indicate an optional parameter and should not be typed.

Example : clearz n1 [, n2 to n3] means clearz n1
and clearz n1,n2 to n3

2) - All the variables can be replaced by litteral values in all instructions.

Example : n1=1:n2=10:n3=15

```
clearz n1,n2 to n3
```

is equivalent to

```
clearz 1, 10 to 15
```

IMPORTANT :

THE MASTER DISK HAS 'WRITE PROTECT LABELS' INSTALLED WHICH WILL PREVENT THE USE OF THE DISK FOR ANY MORE STORAGE WHICH COULD RESULT IN THE LOSS OF 'MASTER' MAIN PROGRAMS. DO NOT REMOVE THE WRITE PROTECT LABEL. TO DO SO MAY CAUSE YOUR PROGRAM TO BECOME UNUSABLE. AFTER LOADING THE PROGRAM, MAKE A WORKING DISK. (See Appendix D-10).

INTRODUCTION AND DEFINITION

MASTER can be described as an extension to the BASIC interpreter of the Commodore C-64 systems. Written in 6502 machine language to be more efficient, MASTER simplifies all program debugging, screen management (data acquisition and display), printer management and file management. In fact, MASTER improves the work of the three peripherals -the screen, the floppy disk unit, and the printer- for easier and faster results.

SCREEN GENERATOR - MASTER SCREEN

The screen generator was developed to make screen acquisition programming easier. It offers a reliability and a rapidity for data acquisition not available with BASIC routines.

The display can be done inside pre-defined zones allowing complete control or simply, by giving (x,y) coordinates of the start of display.

PRINT GENERATOR - MASTER PRINT

MASTER print generator permits optimization of all printing, creation and debugging being similar to MASTER SCREEN.

With powerful instructions, similar to MASTER SCREEN instructions, it can also save print masks and pages on disks which can be called from programs. Thus saving time, run time and memory.

FILE GENERATOR - MASTER FILE

MASTER is designed to create and maintain files of records on floppy disks using **ISAM** type indexed files.

MASTER permits entry, query, update, and deletion of a record given its access key. These functions are performed in real time. So, a MASTER file is always sorted in the key sequence as soon as a new record is inserted.

File management is particularly optimized in order to access a record rapidly thanks to MASTER's internal logic and to the operations that are at the developer's disposal.

Also included in the MASTER FILE chapter is data packing/unpacking that optimizes disk storage.

Assuming that your system is equipped with a software transparent interface allowing you to connect your C-64 to any IEEE Commodore disk drives, MASTER FILE is compatible with all these disk drives from 2031 to 9090. By default, the system assumes that a 1541 single disk drive is used.

NOTES : The data packing is primarly to be used with the MASTER FILE instructions, but it can also be used for reducing the size of variables inside the memory.

BASIC COMPLEMENT - MASTER BASIC

As a complement to the three I/O generators already briefly described, programming and control aids are included within MASTER.

Very useful in the design of applications are :

- multiprecision arithmetic (22 digits)
- automatic date validity control,
- computed GOTO and GOSUB,
- simplified random access commands,
- character search within a string.

All these instructions will be fully defined and described later.

EDEX - PROGRAMMER'S AID

To complete the MASTER - C-64 ensemble, we have added EDEX, a powerful programmer's aid with 15 new instructions including a TRACE function.

These instructions are dedicated to the debugging of programs, to help the programmer to design faster and more effectively.

MACHINE LANGUAGE MONITOR

Last feature presented is a simple machine language monitor allowing to view the real contents of the memory and the status of the microprocessor or write special routines. Most of the MASTER users will never use the machine language monitor, but advanced programmers will find it very useful.

RELIABILITY-EASIER MAINTENANCE

The maintenance of the applications developed with MASTER is easier, thanks to the controls, in acquisition as well as in file management. In case of accidental destruction of the tables, the utilities enable the rebuilding of the whole table. Besides, MASTER is provided with dump, copy, examination and modification utilities.

PROTECTION OF YOUR APPLICATION

Avoid having programs duplicated and listed, MASTER can offer full protection. An electronic key is available as an option to personalize each application as well as a nolist option.

Two MASTER programs are on the disk. There is a Development version that allow to write, run, save and load programs. It can not be duplicated and must be loaded from the original disk. That version is the one to be loaded when working on a program. (load"m64.boot",8:run)

And there is a Run-time version that allows to load and run programs but does not permit to modify. The LIST is not available. That version can be duplicated and copied as many times as needed. (load"loaderu",8:run).

(For more details, see Appendix G).

INTEGRATION WITH BASIC - BASIC 4.0

MASTER is fully integrated with the C-64 BASIC 2.0 and there should be no problems with assimilation to the developer. To make it even more integrated to other Commodore computers, MASTER includes all BASIC 4.0 commands (see Chapter 9), the version of BASIC used in all 4000 and 8000 serie machines. It means that many applications developped on those previous machines can be very easely transfer to the C-64.

As already stated, MASTER is an extension of the C-64 BASIC interpreter. However, the integration is brought to such a high level that MASTER commands and the original ones merge together. In fact, each instruction is "precompiled", and put in the form of a "token", exactly like BASIC commands. It means that the practice of abbreviating instructions (shifting the second or third letter) is the same for both BASIC and MASTER.

However, be careful, the number of commands being enlarged, the risks of confusion are greater. In the case of confusion, MASTER commands will always take priority. For example, the sC (scR) command will now mean scroll and not scratch (use scrA for the latter). - See Appendix F.

A program that includes MASTER commands will of course be listed correctly only under MASTER. It is the same if you type MASTER commands without being under MASTER : they will not be precompiled. If after loading MASTER you try to run the program, the first MASTER command will generate an error.

In this case, to precompile the commands move the cursor to that line and press **RETURN.**

The similarity with BASIC also applies to the syntactical analysis of parameters which is as flexible as the original BASIC (possibility of including constant data, variables, subscripted variables, complex expressions, etc...).

INSTALLATION AND LOADING

The MASTER package comprises of two elements :

- a disk with the main MASTER program, utilities and demonstration programs.

- a manual

Insert the MASTER disk in your drive, and type:

```
load"*",8  (or load "m64.boot",8)      and 'RETURN'
run        and 'RETURN'
```

At the end of this execution (loading time : 1' 30''), the following messages appear. (See Notes)

```
*** master64 - v1.1 ***
(C) micro-application

dL"bo*
searching for bo*
loading
run

ready.
```

As soon as MASTER is loaded in the computer, it loads and runs a program called **boot**. In your first loading, the program boot will contain only REM statements and a flashing cursor will appear indicating that the system is ready for use. The boot program allows the automatic loading of an application program, or simply the initialization of the computer (color of the screen, border and characters, lower case / graphics, etc...).

Notes :

During the loading of MASTER, it is possible to display a screen-page. It will be done automatically by the loader program if the page is called 'page64'. High-resolution graphic screen pages defined by the program 'THE TOOL' (see your dealer) can also be loaded and displayed. The MASTER 64 logo which is displayed originally by the program in an example of a THE TOOL graphic page.

DISK CONTENTS :

The complete list of all programs on the MASTER disk is as follows :(cA will show the same catalog)

```
0 "master64 v1.1    " 64 2a
4     "m64.boot"          prg
7     "uti64I10d-a"       prg
7     "uti64I10d-b"       prg
3     "loaderu"           prg
70    "master64I10u-a"    prg
54    "master64I10u-b"    prg
7     "uti64I10u-a"       prg
7     "uti64I10u-b"       prg
30    "reserve file"      prg
1     "reserve 1"         prg
24    "reserve 2"         prg
24    "regekey"           prg
46    "copy file"         prg
11    "1541 backup"       prg
1     "boot"              prg
10    "demoprg"           prg
5     "demoscreen"        prg
21    "proscreen 64"      prg
8     "help"              prg
37    "page64"            prg
95 blocks free.
```

It is recommended that the disk be backed up as soon as possible and used only to load MASTER development program.

After loading the MASTER main program, use the 1541 BACKUP program to generate a working copy of MASTER. In the catalog of that new disk, the program M64.BOOT won't appear and 291 blocks will be available. (See 2-3 and Appendix G).

DISK CONTENTS (continued) :

m64.boot	: Loads / initialises MASTER dvt version
uti64Il0d-a	: Utilities - Instructions type UTI*** (See Appendix D)
uti64Il0d-b	: ----------
loaderu	: Loads / initialises MASTER run-time version
master64Il0u-a	: First part of MASTER run-time version
master64Il0u-b	: Second part of MASTER run-time version
uti64Il0u-a	: Utilities for run-time version Instructions type UTI*** - (See Appendix D)
uti64Il0u-b	: ----------
reserve file	: Definition of a MASTER FILE (See Chapter 6)
reserve 1 reserve 2	: Programmed utilization of RESERVE FILE : (See Appendix D-7)
regekey	: Restructures a MASTER FILE (Appendix D-8)
copy file	: Tool to copy MASTER FILE (See Appendix D-9)
1541 backup	: Single disk drive backup program. (See Appendix D-10)
boot	: Program loaded automatically by MASTER after its loading.
demoprg	: Demonstration of ISAM / MASTER FILE (See Appendix E)
demoscreen	: Screen management demonstration. (See Appendix E)
proscreen 64	: Screen-page generator (Appendix E)
help	: Help screen-page for proscreen 64
page64	: High-resolution logo for MASTER 64

COMPATIBILITY

MASTER is available on different Commodore computers including 4032, 8032, 8096, 8032 and Z-RAM board, C-64 and the B-series. THE TOOL, a similar program with extra graphic capabilities is also available for the C-64.

The following board shows the compability between all the MASTER versions for those Commodore users who have been using other CBM machines.

Instruction Compatibility	MASTER SCREEN	MASTER PRINT	MASTER FILE	MASTER BASIC	EDEX	PM96	BASIC 4.0
MASTER for 4032	* (1)	*	*	*			* (4)
8032	* (2)	*	*	*			* (4)
8096	* (2)	*	*	*	*	*	* (4)
8032-ZRam	* (2)	*	*	*	*	*	* (4)
C-64	* (3)	*	*	*	*		*
B-Serie	* (2)	*	*	*	*	*	* (4)
THE TOOL for C-64	* (3)				*		

Notes :

(1) - 40 columns screen / monochrome
(2) - 80 columns screen / monochrome
(3) - 40 columns screen / colors
(4) - Standard Commodore feature.

Programs saved with the NOLIST option are not compatible from a version to another.

For a complete instruction per instruction comparison, please contact Micro Application.

OVERVIEW

The screen generator integrated into MASTER was designed to help create and store screens. Because it is written in machine language, it is faster, more compact and reliable than similar routines written in BASIC.

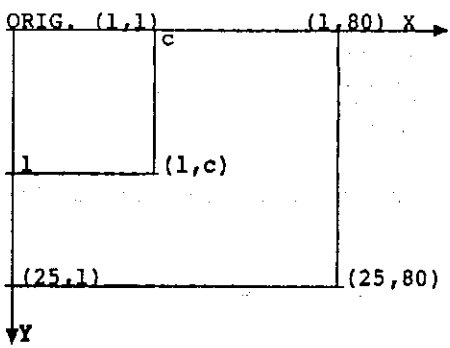
MASTER SCREEN controls the following functions :

- **display commands** to draw lines, columns, to scroll on the screen, etc...,

- **acquisition commands** to define acquisition zones and associated controls, to allocate zone contents to BASIC variables, complete pusing, etc...,

- **screen-page management commands** including saves and loads of screen-page from the disks, screen-page exchanges between screen and memory, etc...,

In the following pages, the names used for variables are arbitrary, they were chosen for easy comprehension. Any BASIC variable would suit, except the reserved variable for MASTER SCREEN "zo".
(see appendices A,B)



The screen is defined as a drawing board with the origin in the upper left-hand corner.
The origin coordinates are 1,1.
Any point on the screen is defined by :
- l = line number
- c = column number

OVERVIEW (continued)

Generally speaking :

- l : line (or row) number (1 <= l <= 25)
- c : column number (1 <= c <= 40)
- ln : length (or number of columns)-horizontal line
- lg : height (or number of lines)-vertical line.

For a better understanding of the chapter, experiment with these commands directly on the screen, in direct or programmed mode.

A **screen page** is a screen layout stored in memory which can be displayed in the computer screen at any time.

It is possible to create complete screen pages with a frame, save them on disk, load them in memory or directly on the screen, fill them with data, and a lot more.

tline - trace line

Function : **tline** draws on the screen a horizontal line defined by its origin and length.

Syntax : **tline** ln, l, c

ln : length of the line,
l : line # of the starting point,
c : column # of the starting point.

Utilisation : The **tline** command is used to draw frames on the screen, or to underline data and titles.

Example : **tline** 40,1,1
tline 40,12,1

These two instructions draw two parallel lines in the centre of the screen.

Notes : To repeat the same instruction with different parameters, use the "←" symbol to avoid writing the body of the instruction each time.

The previous example becomes :

tline 40,1,1 ← 40,12,1

tcol - trace column

Function : **tcol** draws a vertical line on the screen defined by its origin and length.

Syntax : **tcol** lg, l, c

lg : height of the vertical line
l : line # of the starting point
c : column # of the starting point

Utilisation : The **tcol** command is used to draw frames (with tline)

Example :
```
tcol 12,1,1
tcol 12,1,40
```

These two instructions draw two parallel columns in the upper part of the screen.

Notes : 1) Similar to tline, "←" symbol simplifies the coding.

The previous example becomes :

```
tcol 12,1,1 ← 40,12,1
```

2)
```
10 tline 40,1,1 ← 40,12,1
20 tcol  12,1,1 ← 12,1,40
```

These two BASIC lines draw a frame. Notice that the intersections of the frame are automatically formed.

clear

Function : **clear** clears the screen from a given location.

Syntax : **clear** ln, l, c

ln : length (or # of characters) to clear
l : line # of the starting point,
c : column # of the starting point.

Utilisation : **clear** is used to clear parts of the screen. The command clears only the parts of the display defined, not other parts.

Examples : To clear the first line of the screen

clear 40,1,1

To clear the frame already drawn and also all that's inside the frame

30 for i = 1 to 12 : **clear** 40,i,1 : next i
(clears a "window" of 12 lines and 40 columns).

Notes : "←" symbol can be used (see tline)

Ex : **clear** 10,1,1 ← 10,2,2

out

Function : **out** displays on the screen a string, a$ at a given location

Syntax : **out** a$,l,c

a$: any string
l : line # of the starting point
c : column # of the starting point.

Utilisation : **out** displays a string on any part of the screen without having to use the cursor keys in string form.

out also facilitates frame filling.

Example :
```
40 a$ = "JOHN SMITH"
50 out a$,5,10
```

This two line program will display JOHN SMITH on the fifth line starting at the tenth column.

Note : "←" can be used (see tline)

```
40 a$ = "JOHN SMITH" : b$ =  "PATTY DAVIS"
50 out a$,5,10 ← b$,5,25
```

scroll - **scroll** part of screen

Function : **scroll** shifts, up or down, a part of the screen or window. It is defined by its starting point, its length and height.

Syntax : **scroll** l,c,lg,ln,ty

l : line # of the starting point
c : column # of the starting point
lg: height of the window (or number of lines)
ln: length of the window (or number of columns)
ty: type of scroll - u for up
- d for down

Utilisation : **scroll** allows display of files beyond normal line capacity. With simple coding, it is possible to scroll data in a frame and as soon as the value is found, to save it and continue with the program.

Example :
```
10 sclear:                          rem   screen clear (see 4-22)
20 a$ = "JOHN" : b$ = "PATTY" : c$ = "JULIE"
30 out a$,11,4 ← b$,11,10 ← c$,11,17
40 for i = 0 to 9
50 scroll 10-i,4,2,4,u
60 clear 4,11-i,4
70 scroll 11+i,17,2,5,d
80 clear 5,11+i,17
90 for j=1to200:next j:      rem reduces program speed
100 next i
110 end
```

Just type the program and see what happens. You can also run the program without the lines 60 and 80 to see the difference.

Note : "←" Can be used (see tline)

- A **scroll** does not erase the previous output.

rev - reverse part of screen

Function : **rev** turns an area of the screen (ie. a window) to another color. The area is defined by its starting point, length height and color.

Syntax : **rev** l,c,lg,ln [,co]

l : line # of the starting point
c : column # of the starting point
lg: height (or number of lines)
ln: length (or number of columns)
co: new color optional
(0 to 15 - See next page)

Utilisation : **rev** draws the user's attention to a section of the screen. By using the instruction more than once with the same coordinates, it is possible to make the window blink. The speed of the blinking can be regulated by the use of a timing loop (ie. for : next).

Example : To display two names and use the reverse mode on those displays. :

```
140 a$ = "JOHN" : B$ = "PATTY"
150 out a$,5,9 ← b$,8,8
160 rev 5,9,1,len(a$) ← 8,8,1,len(b$),4
```

Note : "←" can be used (see tline)

screen : screen color definition

Function : screen allows the modification of the colors of the screen, the borders and the characters displayed by MASTER.

Syntax : screen sc,br[,cr]
with sc: color of the screen (0<=sc<=15)
br: color of the border (0<=br<=15)
cr: color of the characters (0<=cr<=15) optional

Utilization : Using the screen instruction, you can easily define the colors of your screen, border and characters displayed by MASTER (using tline, tcol, out, outz and reqz - see these instructions).

If the color mode is BLUE for the characters, when the 64 is turned on, the modification of this color using a SCREEN will be effective only for the characters displayed by MASTER instructions.

Examples :

```
10 rem :::: let's make some national flags::::::::::::
20 sclear:screen 1,6,2:out"U.S.A",12,20:gosub100
30 sclear:screen 1,5,2:out"ITALY",12,20:gosub100
40 sclear:screen 0,2,7:out"GERMANY",11,7:gosub100
50 sclear:screen 1,2,2:out"JAPAN",11,9:gosub100
60 end
100 for i=0to500 : next : return
```

Note : 1) **Color code table**

Parameters	Colour	Code
FOR SC BR and CR	Black	0
	White	1
	Red	2
	Cyan	3
	Purple	4
	Green	5
	Blue	6
	Yellow	7
	Orange	8
	Brown	9
	Pink	10
	Grey 1 - Dark	11
	Grey 2 - Medium	12
	Light Green	13
	Light Blue	14
	Grey 3 - Light	15

2) STOP and RESTORE will give back the original colors, black screen and white characters.

MASTER SCREEN
DATA ACQUISITION INSTRUCTIONS

Overview :

MASTER acquisition instructions have been designed to be used instead of INPUT and GET to allow homogeneous, controlled, efficient and reliable data acquisition.

This section is dedicated to MASTER acquisition commands. A zone is defined by certain parameters, location, size, type of characters allowed, other possibilities of exit (than return) and an identification number. **decz**

Once the zone is defined, the **reqz** command instructs MASTER to bring data in, perform the controls assigned to this zone and produces the screen control functions, such as moving the cursor, deleting, inserting. The **inz** command performs information transfer from a zone to a string, **outz** performing the opposite operation. 128 different zones can be defined on one page or screen.

BASIC	MASTER
input a$	**decz** 1,... : zone definition **reqz** 1 : controlled acquisition **inz** 1,a$: transfer into a$

See more details in the following pages.

decz declaration zone

Function : **decz** is used to define all parameters of a screen zone for controlled acquisition.

Syntax : **decz** n,l,c,ln [,ty] [,f$]

```
n  : identification number          (0 <= n <= 127)
l  : line # of the starting point   (1 <= l <= 25)
c  : column # of the starting point (1 <= c <= 40)
ln : total length of the zone       (1 <= ln <= 254)
```

ty : type of zone control
value of ty :
ty = n numeric data only,
ty = m data in capital letters,
ty = r other possibilities of exit than with return,

* for a numeric zone : all the keys, numeric ones excepted,
* for an alphanumeric zone : all non-alphanumeric keys.
* The ASCII code of the key you pressed for exit is put in zo, so you can define and use control keys.

```
F1 to F8       : zo = 133 to 140
CRSR DOWN      : zo = 17
CRSR UP        : zo = 145
SHIFT/RETURN   : zo = 141
CTRL / A to Z  : zo = 1 to 26
```

ty = p zone defined in print using + f$
f$: string containing the format for the print using.

A format can define a zone. When acquiring or displaying data in this zone, its contents are formatted. The format is stored in a string (here : f$). The control characters are :

```
9 : numeric,
8 : numeric; set to zero if nil
1 : sign position  (only -)
2 : sign position  (with + or -)
```

Any alphanumeric character can be inserted into a format.

Ex : f$ = "$ 19,999,998.88"

Utilisation : **decz** allows you to define, as precisely complete as possible, the acquisition zones. It is now possible to work efficiently on a zone by using its identification number.

decz

Example : decz 2,1,2,10,n,p,"199 999.88"
This declaration defines a zone (number 2) starting at (1,2) of 10 characters. Only numbers will be accepted (n). Printusing format is used with impression of the sign if it is minus and zero being forced at the end of the number. The number will be 5 digits long with 2 digits after the decimal point.

Notes : "<-" can be used (see tline)

- More than one type can be combined in the same decz. There are all these different possibilities.

n	: Only numbers	- RETURN to leave the zone.
n,r	: Only numbers	- A programmable key to leave the zone.
n,p	: Only formatted numbers	- RETURN to leave the zone
n,r,p	: Only formatted numbers	- A programmable key to leave the zone.
m	: Only capital letters	- RETURN to leave the zone
m,r	: Only capital letters	- A programmable key to leave the zone.
r	: Every character accepted	- A programmable key to leave the zone.
p	: (equivalent to n,p)	
r,p	: (equivalent to n,r,p)	
(nil)	: every character accepted.	- RETURN to leave the zone.

If the data overflows the acquisition zone, the zone is highlighted and an error beep is generated. The cursor moves back to the beginning of the zone . This prevents leaving the zone if the format is not correct (the format is controlled when you press RETURN).

If you try **decz** instructions without using a reqz after it, nothing will happen but the declaration will remain.

reqz requisition zone

Function : **reqz** performs the acquisition using the parameters defined in the corresponding decz.

Syntax : **reqz** n

n : zone identification number

Utilisation : **reqz** is used in association with decz. If a corresponding decz does not exist (or has not been loaded in a screen-page), an error message will be generated.

With this command, MASTER takes the data acquisition under its control, as defined in the zone declaration. On execution of this command a cursor is displayed at the beginning of the zone.

The CLR, HOME, CURSOR BACK and FORWARD, INST/DEL functions can be used in the zone.

The data acquisition is controlled as defined in the zone declaration.

If the mode used to define the zone is r, and the exit code is different from 13 [ASCII value for RETURN] (zo), the cursor will return to that position occupied on exit from zone if this zone is re-entered.

Example : If the previous example has been entered (decz 2,...), just type

reqz 2

to see what happens. Only numbers will be accepted.

Notes : "←" is allowed (see tline), but if more than one decz of r type (using a programmable key to leave the zone) is used, then something like reqz 1 ← 2 ← 3 cannot be used, because the value of zo cannot be tested after each **reqz**.

inz in memory zone

Function : inz transfers the content of a zone to a string when all data in a zone have been controled.

Syntax : inz n,a$

n : zone identification number
a$: any string variable.

Utilisation : inz is used to transfer zone contents to a string.

Example : inz 2,a$
If the two previous examples have been tried, this one should have transfered the contents of the zone into a$. To verify, just type

```
out a$,3,5
```

```
10 decz 1,25,10,5,r
20 reqz 1 : inz 1,a$
30 print tab(20) zo,a$
40 goto 20
```

Notes : "←" is possible (see tline)

The string variable receiving the contents of a zone will have the same length as the zone, whatever the contents of the zone are.

outz out to zone

Function : **outz** displays a string in a zone.

Syntax : **outz** n,a$
n : zone identification number
a$: any string variable.

Utilisation : **outz** is the opposite of inz. Data goes from a variable to the zone. This instruction is similar to the display instruction out but is easier to use and more precise.

Easier because the zone number instead of the coordinates is required.

More precise because it assumes all the controls defined in the zone by the decz.

Example : still using the previous examples
a$ = "11111"
outz 2,a$

Notes : "←" still available (see tline)

In the case of a zone defined in print using if an overflow of decimals occurs the number is truncated.

If an overflow of capacity occurs, the zone is filled with"*****" and an error beep is generated.

In the case of an **outz** in a zone declared with no sign (print using without 1 or 2), of a string with a +,- or blank as a first character, the string can be shortened by a digit, or the display can be replaced by stars. To avoid this, extract the sign by :

a = len(mo$) - 1 : mo $ = right$(mo$,a)

or using getst (see 6-31)

getst mo$;p,mo$,a,2

clearz **clear zone**

Function : clearz clears a zone.

Syntax : clearz n1 [,n2 to n3] [,...]

n1,n2,n3 zone identification numbers.

Utilisation : clearz can be used to "clear" the contents of a frame without clearing the frame itself, after an acquisition and before a new acquisition.

Example : clearz 2

revz - reverse video mode in zone

Function : **revz** reverses the video in a zone.

Syntax : **revz** n1 [,n2 to n3] [,...]
n1,n2,n3 zone identification numbers.

Utilisation : **revz** can be used to bring the user's attention to the contents of a zone, for a result or a message. It is possible to blink the zone (see rev 4-8).

Example : **revz** 2

SCREEN PAGE MANAGEMENT INSTRUCTIONS

OVERVIEW

In a business application, the data acquisition frames are often similar and are used more than once, with different information. In a normal BASIC program, a new frame has to be created each time, wasting both time and memory.

MASTER defines a **screen page**, with all the decz (declarations), and saves that page on disk. Avoiding the necessity of creating a frame over and over. Different applications can have their own screen pages. Coding is simplified.

The screen page can be loaded in memory, worked on, filled with data, and then display the page already prepared. It is possible to exchange an actual screen page with a page in memory, for example, a HELP screen.

If during an acquisition, the control commands are forgotten, the program can easily exchange pages for a HELP page, read the commands, and again exchange the pages. The cursor will reappear automatically at the exact spot it was occupying before the page exchange.

The screen management is performed by the following seven instructions:

sset screen **set**

Function : **sset** defines the screen page about to be worked on (current screen page).

Syntax : **sset, sset** pm
pm : screen page id number (0 <= pm <= 127)

Utilisation : In the absence of this command, the current page is the one on display. Therefore, every MASTER SCREEN instruction will be done on the screen. If **sset** pm is used, the display and acquisition instructions will control the screen page in memory. Therefore, nothing will be executed on the screen.

To display pm page on the screen use scopy or sexch (see these instructions).

To return to the screen without modifying the actual screen page, just use **sset**.

Example : **sset** 1 : sclear
tline 40,1,1 ← 40,12,1
tcol 12,1,1 ← 12,1 40
sset

This just drew a board (see 4-4) on page 1 but nothing appeared on the screen. Typing :

tline 40,1,1

will draw a line on the screen.

Note : Before working on a page it must be cleared (or initialized) using the sclear instruction (see 4 - 22)

scopy screen **copy**

Function : scopy transfers a page from memory to the screen.

Syntax : scopy

Utilisation : After having defined the page to be transfered using the sset pm instruction, **scopy** will display page pm including any zones.

Example : To display page 1 previously defined

```
sset 1
scopy
```

The board should appear.

Note : Typing

```
tline 10,5,5
```

The line will not appear on the screen but will be drawn in memory on page 1.

Typing

```
sset : tline 10,5,5
```

The sset instruction will now permit screen editing and the tline will be executed on the screen.

sexch - screen **exch**ange

Function : sexch exchanges the current screen page with one in memory.

Syntax : sexch

Utilisation : sexch is used to display a page in memory on the screen and to return later to the original screen page. This can be very useful to consult HELP pages, or data files, etc.. The sset pm instruction should be used before **sexch** to define the page in memory.

Example :

```
sset
tline 22,1,1 <- 22,12,1
sset 1
sexch
```

The original frame is on display

```
sexch
```

The two previously drawn lines return.

sclear - screen **clear**

Function : sclear clears a page on the screen or in memory.

Syntax : sclear

Utilisation : sclear must be used with the sset pm instruction. It clears the page of data but leaves the zone declarations.

Example :

sset : **sclear**

will clear the screen

sset 1 : **sclear**

The frame previously defined on page 1 is erased but still exists.

Note : The memory space used by a page is not freed when a sclear is done. The memory is still reserved for the page (see sreset)

sreset - screen **reset**

Function : **sreset** will reset a memory page.

Syntax : **sreset** pm
pm : memory page id number.

Utilisation : **sreset** is used when a page in memory is no longer required. **sreset** clears the page and the zone declarations. The memory is now available to BASIC.

Example :
```
sset 1                  :rem creation of page
tline 5,5,5             :
tcol 5,5,5              :
scopy                   :rem and display of page
sset 1                  :
sclear     :  ? FRE(0)
sreset 1   :  ? FRE(0)
```

Note : The difference between the two displays should be 1000 this corresponds to memory taken by a page. This is also the same as the screen (40x25) memory.

The line and column drawn may still be in memory, but that memory is now available to BASIC.

ssave - screen **save**

Function : **ssave** saves a page in memory on disks.

Syntax : **ssave** du, "[dr :] name " [,nl] [,n2 to n3] [,..]
du : disk unit device number (8)
dr : drive number (0 or 1) - 0 :default value.
nl,n2,n3 : associated zones

Utilisation : if no sset pm is used, the current screen page will be saved. If sset pm is used, the memory page number pm will be saved. The zones (decz) can also be saved, using the syntax, nl or , n2 to n3. This syntax is optional. In all cases, all that is on the page will be saved (frames, boards, data or display...)

Example : **ssave** 8,"prgl",1,12,30 to 70

Note : **ssave** 8,"name",0 to 127 would save all associated declarations. There are no errors if non-declared zones are saved.

sload - screen load

Function : **sload** loads in memory or on the screen a page saved on disk.

Syntax : **sload** du," [dr:] name"
du : disk unit device number (8)
dr : drive number (0 or 1) - 0 : default value

Utilisation : Same principle as ssave.
If sset pm is used, the loaded page becomes the page in memory number pm. If sset is current, the page will be displayed on the screen. All the zones will be declared again.

Example :

```
sset
sload  8,"prg1"
```

OVERVIEW

MASTER PRINT is a powerful print generator that brings to printing the same features MASTER SCREEN brings to screen management (acquisition excepted).

It defines the print layout.

The principle of use is as follows :

First, define a page on the disk using the **pcreate** instruction, this defines the general format of the page (margins, blank lines before/after printing, etc...)

Then, in this pre-defined page, exactly as with a memory page, declare display zones similar to MASTER SCREEN. (with **pout, pclear, pdecz** instructions...).

Lastly, start printing the page. (**pprint** instruction).

The printing program just sets the data into the predefined zones. If, for instance, the preprinted paper is changed, only the page definition has to be modified, and not the print program.

Note : (Advanced Programming) - A page is composed of a relative file containing a record per line (the record being a line long) + one record per zone. When printing, all the information is displayed on this page (this permits printing the lines in any order.); then, the page is sent to the printer.

GENERAL ORGANIZATION OF A PRINT PROGRAM

1 Creation of a print page

pcreate

2 Definition of the page

popen	Opening the page,
pdecz	Zones declaration,
pout	Headlines and frames definition,
pclose	Closing the page.

3 Filling a print page

popen	Opening the page,
pclearz	Clearing of the zones (if needed)
pout	Filling the page/zones with data,
poutz	
pclose	Closing the page.

4 Printing of the print page

popen	Opening the page,
open n,4	Opening a printer,
pprint n	Printing of the page,
close n	Closing the printer,
pclose	Closing the page.

Notes : Points 3 and 4 can be repeated for the display of the same pages with different sets of data.

Points 1 and 2 can be done out of the application programs (to save memory),

Points 2 , 3 and 4 can be grouped together using only one **popen** and one **pclose** instead of three and three.

pcreate - print page **create**

Function : **pcreate** defines the parameters of a print page,and saves them on disk.

Syntax : **pcreate** du, "dn:"+np$,nl,nc,hp,fp,mr
du : floppy device number (8)
dn : drive number (0 or 1) - 0 :default value
np$: page name (16 characters maximum)
nl : number of lines (between 1 and 255)
nc : number of columns (between 1 and 255)
hp : # of blank lines at top of page (between 0 and 255)
fp : # of blank lines at bottom of page (between 0 and 255)
mr : left-hand margin (between 0 and 255).

Utilisation : **pcreate** is the first instruction used when defining a new print page. It is an independant instruction and does not need other complimentary instructions (to access the disk for example, or popen and pclose).

Example : To create a print page called page 1 with 60 lines, 132 columns, with 3 blank lines before, 3 blank lines after printing and a left margin of 10 characters, the instruction is as follow :

pcreate 8,"page1",60,132,3,3,10

The print page will be saved on drive 0 of unit 8.

Note : **pcreate** must be used only once for every print page.

popen - print page **open**

Function : **popen** opens a print page.

Syntax : **popen** du,["dn:"+] np$

du : Floppy device number (8)
dn : Drive number (0 or 1) - 0 : Default value
np$: page name.

Utilisation : With the pcreate instruction, a general format for the print page has been defined. **popen** reads the format of the predefined page before any use. If this page has already been used, with zone declarations, those declarations will also be read with a **popen**.

Example : **popen** 8,"page1"

Notes : 1 The closing of a print page is mandatory after any work on a page and it is done by the pclose instruction (see 5-5)

2 Only one print page can be open at a time.

3 During a **popen**, the management of all disk access is done automatically by MASTER.

pclose - print page **close**

Function : pclose closes a print page

Syntax : pclose

Utilisation : pclose is used after any work on a print page (but pcreate). It must be used after the detailed definition of a page, the filling of a page and the printing of a page. **pclose** is a mandatory instruction. It saves on disk the last version of the page with all zone declarations, and the zone values, if any.

Example : popen 8,"page1"
(any work on the page (pout, pdecz...))
pclose

Note : Forgetting to use **pclose** can result in the loss of data from the page.

pdecz - print page declaration zone

Function : **pdecz** declares a zone for controlled data acquisition that is to be printed

Syntax : **pdecz** n,l, c, ln [,ty] [,f$]

n : zone id number (from 0 to 127)
l : line # of starting point (bewteen 1 and 255)
c : column # of starting point (between 1 and 255)
ln: total length of the zone (between 1 and 255)
ty: type of control associated to the zone
n - numerical
m - capital letters
p - print using
f$: print using format (for p-type)
(see decz : 4-11,4-12).

Utilisation : The **pdecz** instruction has the same structure as the MASTER SCREEN decz instruction. In fact, **pdecz** is used the same way as decz with the following limitation. The MASTER PRINT instructions do not really acquire data, but control them. The type ty : r (like request) does not exist. The other types can be used. In the case of overflow or an unacceptable character type, error messages (similar to MASTER SCREEN message,) will be generated.

Example : **pdecz** 2,10,12,20,m - corresponds to a zone id = 2, at the 10th line and 12th column of the page, with a length of 20 characters with only capital letters.

Notes : 1) **pdecz** can be used only if a page has been opened (popen).

2) See decz 4-11,4-12 for more information.

pout - print page **out**

Function : pout prints a string of characters in a page at a given location.

Syntax : pout a$,l,c
a$: string of characters
l : line # of starting point
c : column # of starting point

Utilisation : pout similar to out in MASTER SCREEN.
(See out a$,l,c 4-6)

Example : pout "PRINT MANAGEMENT",20,20

.....

poutz - print page **out** to **zone**

Function : poutz prints a string of characters in a zone of the page.

Syntax : poutz n,a$
n : zone id number.
a$: string of characters.

Utilisation : poutz similar to outz in MASTER SCREEN.
(See outz n ,a$ 4-15)

Example : poutz 2,"MANAGEMENT"

.....

Note : pout and **poutz** can be used only if a page has been opened (popen).

pclear - print page **clear**

Function : **pclear** clears a part of a line in the page.

Syntax : **pclear** ln,l,c
ln : length or # of characters to be cleared
l : line # of starting point
c : column # of starting point

Utilisation : **pclear** similar to clear in MASTER SCREEN

Example : **pclear** 19,20,20

.....

pclearz - print page **clear** zone

Function : **pclearz** clears a zone in the page.

Syntax : **pclearz** nl[,n2 to n3][,...]
nl,n2,n3 : zone id numbers.

Utilisation : **pclearz** similar to clearz in MASTER SCREEN

Example : **pclearz** 2

.....

Notes: 1) **pclearz** 0 to 127 will clear all the zones. This is very useful when a print page has to be printed more than once with different information. (see 5-2,5-3)

2) **pclear** and **pclearz** can be used only if a page has been opened. (see 5-2).

perase - print page **erase**

Function : perase erases a print page completely.

Syntax : perase

Utilisation : perase is used inside the cycle popen-pclose. It destroys a print page but keeps the format defined by pcreate. It can be used when a page is no longer required but is to be replaced by another one with the same general format.

Example :
```
popen
instructions pdecz, pout, poutz....

pprint
perase

instructions pdecz, pout, poutz....

pprint
pclose
```

NOTES :

1) **perase** can only be used after a popen.

2) a print page destroyed by **perase** can not be loaded with popen.

pprint - print page **print**

Function : **pprint** prints a print page.

Syntax : **pprint** np
np : channel number.

Utilisation : **pprint** is used in the following way, after having filled the page with data and having opened (popen) the page, before the print command is given a channel for the printer must be opened.
The rest is MASTER's work.

Example :
```
open 1,4                : rem  the page will be printed
pprint 1                : rem on the printer
close 1

open 1,4                : rem the page will be printed
pprint 1 : pprint 1     : rem twice.
close 1
```

Notes : 1) **pprint** can only be used after popen.

2)
```
open 1,3
pprint 1
close 1
```

In this last example, the print page will be printed on the screen (peripheral number : 3). This is a useful check before printing.

OVERVIEW

Another kind of printing is offered by MASTER. In the case of a large table containing data, identical in presentation but different in value, it is possible to use the BUFFER ZONE. Used mainly in file management, this buffer can be used to declare one line corresponding to a print line. All the decz, clearz, out, outz instructions (of MASTER SCREEN, but not MASTER PRINT) are available with a slightly different syntax (instead of l, line starting point : b must be used).

General Organisation of the Instructions

```
        open -,4                  Opening of a printer channel
        deczl,b,...               Declaration of the buffer-zone

LA  :   clearzl                   Initialisation of the buffer-zone
        decz                      Declaration of sub-zones,
                                  inside the buffer-zone
        outz instructions         Filling of the sub-zones
        inz l,a$                  Transfering all data from the
                                  buffer to a variable
        print #-,a$               Printing of the variable
        go to LA                  Loop
```

Note :
In this case, b is not a variable but part of the syntax indicating the use of a buffer zone.

Example

```
list
 10 dim a$(20)
 20 open 1,4:                    rem open printer channel
 30 decz 0,b,1,80:               rem buffer (b) zone declaration
 40 :                            rem starting point:1  length:80
 50 clearz 0:                    rem initialise buffer zone
 60 for i=1to4
 70 decz i,b,i*15,10:            rem declare 4 sub zones of length 10
 80 next i
 90 for i=1to20
 100 a$(i)=str$(i*rnd(0)):       rem data creation
 110 next i
 120 for j=1to5
 130 for i=1to4
 140 outz i,a$(i+(j-1)*4):       rem data transfer to sub zones
 150 next i
 160 inz 0,b$:                   rem transfer of main zone into b$
 170 print#1,b$:                 rem printing of line
 180 clearz 0,b:                 rem erasing of buffer zone
 190 next j
 200 close 1
 210 end

ready
```

Notes : 1) Because there is no disk activty this program will run in a short time. This is useful when doing repetitive jobs.

2) The use of a clearz for the buffer is mandatory to clear it as it is used by the file management system and random characters may be added to the print out.

3) This buffer zone can also be used to save data during an overlay of programs.

```
10 rem master print packed values in c-type
20 rem (see output 6-33)
30 rem
40 dim comp(255),comp$(255)
50 for i= 1 to 255
60 comp(i)= i -int(i/4):comp$(i)=right$(str$(comp(i)),3)
70 next i
80 open 1,4
90 creatst ho$,80,"-":ve$="I"
100 pcreate 8,"1:alphacomp",10,80,2,0,0
110 popen 8,"1:alphacomp"
120 pout ho$,1,1 ← ho$,6,1 ← ho$,10,1
130 for i = 2 to 10
140 pout ve$,i,1 ← ve$,1,80
150 next i
160 for i = 7 to 9
170 pout ve$,i,17 ← ve$,i,33 ← ve$,i,49 ← ve$,i,65
180 next i
190 pout"ALPHANUMERIC PACKING C-TYPE, PACKING VALUE TABLE.",3,5
200 pout"N - ORIG. STRING LENGTH / lc - PACKED LENGTH ",4,5
210 for i = 0 to 4
220 pout "N lc",8,5+16*i
230 next i
240 pprint 1: rem title
250 pclose
260 j = 1
270 rem advanced programming buffer zone b
280 rem
290 decz 0,b,1,80
300 clearz 0
310 for i = 0 to 4
320 decz 2*i+1,b,3+i*16,3,p,"999"
330 decz 2*i+2,b,10+i*16,3,p,"999"
340 next i
350 for i = 0 to 4
360 out ve$,b,i*16
370 next i
380 out ve$,b,80
390 for i = 0 to 4
400 a=51*i+j:a$=right$(str$(a),3)
410 outz 2*i+1,a$
420 outz 2*i+2,comp$(a)
430 next i
440 inz 0,a$: rem transfer buffer zone to a$
450 print#1,a$: rem printing line
460 clearz 0
470 if j = 51 then 500
480 j = j + 1
490 goto 350
500 print#1,ho$
510 close 1
520 end
```

MASTER FILE
FILE MANAGEMENT SYSTEM

OVERVIEW

MASTER FILE is a new type of file management for all COMMODORE disk drives. The MF (abbreviation for MASTER FILE) file management has its origin in the Indexed Sequential Access Method (ISAM). MF was designed for the Commodore range of computers. It offers more possibilities than the traditional Isam or the BASIC 2.0 or 4.0 original file management, as well a RAPIDITY and RELIABILITY essential in management applications. Emphasis must be placed on the fact that **the file definition function, file creation and utilisation functions, are distinct in MF**, for a better flexibility, and memory saving.

File definition : The definition of the files is done by a special program on the MASTER disk. RESERVE FILE (see 6-2). During the file definition, the programmer defines the record length and the record field which will become its access key. MF reserves on the disk the requisite space for the index file. However, the MASTER File itself is not reserved and the space is allocated only during the utilisation of the file. Disk space management, disk allocation for new records and index management are totally transparent for the programmer, the application and the end user.

File creation and utilisation : Once the file is defined, MF manages the creation and utilisation of the files. MF allows entry, query, update and deletion of records using its access key. These functions are performed DYNAMICALLY because the indexes are updated in real time. For this reason, an MF file is always sorted according to the key sequence as soon as you insert a new record.

Record access method : With MF, there are six ways to access a record. This allows an exhaustive and quick exploitation of a file. These methods are :

- strict retrieval (complete key)
- retrieval from a radical (incomplete key)
- extraction according to a retrieval mask
- previous or next (-/+) record following the key order
- -/+ record with a retrieval mask
- reading in the order of creation.

It is possible to combine these different methods. The enter, update and delete commands are still available while one of the other methods is being used.

Security : In case of accidental destruction, an utility is ready to rebuild the index, thus maximum security. (see Appendix D REGEKEY)

Overview :

The RESERVE FILE program is used to create a MF file. Of course, it will only work under MASTER's control. MASTER must be loaded first because of this.

To illustrate all explanations during this chapter, and specially the definition of the file, try creating this example of a file. A hard-copy of the screen during the utilisation of RESERVE FILE is included in this document (see 6-3) and will facilitate the comprehension.

STAFF FILE :

Each record represents the card of an employee, containing the following informations :

- Social Security number	: 13 characters
- First name	: 18 characters
- Last name	: 18 characters
- Address	: 70 characters
- Telephone number	: 8 characters
Total	127 characters

```
                ISAM  File  Reservation
+-------------------------------------+
!  file name                : employee  !
!  drive for records        : 0         !
!            keys           : 0         !
+-------------------------------------+
!  level                    : 1         !
+-------------------------------------+
!  total number of records  : 500       !
+-------------------------------------+
!  record length (with key): 127       !
!  key length    (1<=n<=30): 13        !
!  key position             : 1         !
+-------------------------------------+
+-------------------------------------+
!  memory table length      :  943      !
+-------------------------------------+
!  total record blocks      :   251     !
!  total index blocks       :   52 +  4 !
!  total number of blocks   :   307     !
+-------------------------------------+
   ok (y/n/h):y      creation (y/n):n
```

```
                ISAM  File  Reservation
+-------------------------------------+
!  file name                : employee  !
!  drive for records        : 0         !
!            keys           : 0         !
+-------------------------------------+
!  level                    : 2         !
+-------------------------------------+
!  total number of records  : 500       !
+-------------------------------------+
!  record length (with key): 127       !
!  key length    (1<=n<=30): 13        !
!  key position             : 1         !
+-------------------------------------+
+-------------------------------------+
!  memory table length      :  223      !
+-------------------------------------+
!  total record blocks      :   251     !
!  total index blocks       :   59 +  1 !
!  total number of blocks   :   311     !
+-------------------------------------+
   ok (y/n/h):y      creation (y/n):n
```

FILE RESERVATION PROGRAM

File name : Any alphanumeric name smaller than 15 characters.

Ex : employee

Drive for records : Either drive 0 or 1. If you are working with a 1541 single disk drive, 0 is the right answer.

Ex : 0

Drive for keys : Similarly, drive (1 or 0) to store the index tables. It is advisable to store file and index on the same disk, but with very large files the index may be stored on a different disk. Again, 0 for a 1541 unit.

Ex : 0

Level : The level is a file design choice.

Level 1 high execution time with a large table in memory

Level 2 slower time with a small table in memory

The choice is made during the running of the RESERVE FILE program (see hard-copy 6-3). After answering the questions concerning the file definition, the program will return the size of the record file, and the size of the index tables.

In the example, level 1 gives a memory table length of 943 bytes, and level 2 gives 223 bytes. Answering **N** to the question OK (Y/N), it is possible to modify the parameters of your file and change for example the level or the key length. The speed loss at level 2 is approximately 30% .

Ex : 2

File reservation program (continued)

Total number of records :

It is important to know (and to indicate to the file reservation program) the maximum numbers of records required. This depends on the size of the records and the space available on the disks.

Ex : 500

Record length (with key) :

Number of characters in each record. The key (or primary index) is an unique field which defines each record, e.g., social security number. Ex : 127

It is possible to optimize the use of the disk space depending on the size of a record. (see optimization 6-7).

Key length : (1 < = n < = 30)

Enter the size of the primary key.
This controls the size of the index tables.

Ex : 13

Key position (byte number) :

Where, in the record, will the primary key be, i.e., character position. In the example, it is the first field so the key position is 1. If the first name had been taken as the primary key, the first byte of the key would have been directly after the S.S. number, i.e., 14.

Ex : 1

Notes : It is also possible to reduce the size of the index tables (see optimization 6-7).

The first byte of the key must have an ASCII value smaller than 250. All alphanumeric characters (in upper or lower case) are acceptable.

File reservation program (continued)

The next numbers are calculated by the program. The four numbers now displayed indicate the space the file will occupy on disk. Answering "N" to the question **ok (Y/N/H)**, allows the modification of the defined parameters. When everything seems to be all right and ready to create the file, just answer "Y" to the question **ok (Y/N/H)** and "Y" to the question **creation (Y/N)**. Answering "H" to the question **ok (Y/N/H)** prints out a hard-copy of the page for your records.

Memory table length

In our example, the memory table length is 943 or 223. (see 6-4)

Total record blocks :

This gives the number of blocks the record file will take on disk (without the index files), 251 in our example.

Total index blocks :

This will give the number of blocks needed by the index file. In the example at level 2, this number (60) is larger than that at level 1, while the index table memory at level 1 is bigger than that at level 2. This corresponds to the fact that at the level 1, only one table manages everything in memory (more memory space taken, with speed) while at level 2, there are two levels of tables; a first one in memory and a second one on disk (less memory space taken with less speed).

Total number of blocks needed :

Important data. If the record file and index file are on the same disk. This number must be smaller than the number of available blocks on the disk to be used.
(caT)

Optimization of the record file

To optimize the use of available space on disk, do not forget that MF does not control sector borders. For example, let's create a file containing records of 32 bytes each. The number of available bytes in one sector is 254. So, the number of records in each sector is : INT (254/32) = 7. Within a space of 254 bytes, 224 (32 * 7) are effectively used, 30 are wasted. It means for a file containing records of 32 bytes, the coefficient of waste is 11.81% (30/254*100). For records of 31 bytes, the coefficient of waste falls to 2.56% since INT (254/31) = 8 ; the waste is now 6 bytes per sector. In the example file, MASTER will store two records per sector, i.e., 254 bytes will be used with no waste of space.

Optimization of the index tables

To optimize the index length, calculate similarly to the above but add 3 bytes (track, sector, pointer) to the key length.

Packing/unpacking of data

To help optimize the use of the disk space choose a low coefficient of waste. The instructions to compress data are comprehensive and will be explained at the end of this chapter (see 6-25).

MASTER FILE/ DOS compatibility

MF files are compatible with BASIC 2.0 and 4.0. This means that the Scratch command can be used and that they can coexist with sequential files, relative files or programs. The copy command does not work with MF files. The collect command **must not be used** with a disk on which an MF file is saved.

Note It is also possible to create directly an MF file from a program. A utility is ready to do this (see the appendix D reserve 1 and 2).

Summary of important data for files

The following facts should be known :
- the file name length cannot exceed 14 characters,
- the record length cannot exceed 254 characters,
- the key length cannot exceed 30 characters,
- the key length must be at least 2 characters,
- a key is a field inside the record,
- it is possible to have the index on one drive and the records on the other (4040, 8050, 8250 drives),
- the record can be packed.

iopen - indexed file **open**

Function : iopen opens a MF file

Syntax : iopen nf,du,"dn:"+fi$
nf : file logical number (between 0 and 9)
du : peripheral number (usually 8)
fi$: file name (14 characters maximum)
dn : drive number (0 or 1)

Utilisation : When opening a MF file, its related index is loaded into memory. To store the index, MASTER takes memory management under control. If there is not enough space inside the memory to store the index, the message "out of memory error" is generated. When you use several files at the same time, the index is managed in the same way. You can open ten MF files at the same time. MASTER can control up to 15, indexes and screen pages, the number of files and pages opened at the same time must not exceed 15.

For a file opening, the MASTER variables have the following values :

- ok = 0 : opening ok
- ok = 128: file not found
- nr: number of records
- ni: number of indexed records
- lr: record length
- lk: key length

Example :
```
1000 rem---------------------- open file
1010 iopen 1,8,"0:employee"
1020 if ok=128 then out "FILE NOT FOUND",1,1
1030 ........
```

Notes : Opening the same file twice without using iclose will generate "file open error". The nr variable gives the number of records entered in the file by iwrite or iadd ; idelete does not modify nr. The ni variable gives the number of records entered in the index, that is to say the records entered by iwrite and ivalidate less those deleted by idelete.

iwrite - indexed file **write** record

Function : The **iwrite** command adds new records to the file and inserts its access key into its related index.

Syntax : **iwrite** nf, en$
nf : File number
en$: complete record string, organized as described when the file was first defined.

Utilisation : When this command is executed the index is updated in real time. It means the record can immediatly be queried, using any access method to do it. The en$ variable which contains the record must be the same length as that defined when the file was created. If it is not, there will be a programming error that will generate the "record format" message.

Be sure that the key access field is well positioned inside the record, otherwise the insertion into the index will not be correct.

The status variables have the following values :
- ok = 0 : execution ok
- ok = 1 : key already defined, execution not done
- ok = 64: file full
- ni: incremented by 1
- nr: incremented by 1

Example :

```
1000 rem------------ insertion of new record
1010 en$=cs$+nm$+pr$+ad$+t1$
1020 iwritel,en$
1030 if ok=1then out"EMPLOYEE ALREADY EXISTS ",10,10
1040 if ok = 64 then out"INDEX FULL",10,10
1050 .....
```

Note : The ni variable represents the number of records in the index. That is to say the records entered by **iwrite** and ivalidate less those deleted by idelete.

iadd - indexed file **add** record

Function : The **iadd** command adds a new record to the MF file but does not insert the access key into its related index.

Syntax : **iadd** nf, en$
nf : file number
en$: complete record string organized as described when the file was first defined.

Utilisation : This instruction is used when records have to be saved quickly (very high execution time of the instruction). Index tables will be updated with an ivalidate (see 6-18), when all data has been recorded (batch method). When executing this command, the index is not updated. So, the record cannot be used (unless you use idata) before the file is validated with ivalidate or iwrite which validates the file.

The en$ variable must have the same length as that defined when the file was created. If not, there will be a programming error that will generate the "record format" message.

The status variables take the following values :
- ok = 0 : execution ok
- ok = 64: file full, execution not effective
- nr: incremented by 1

The nr variable represents the number of records in the file.

Example :

```
1000 rem--------------- add without insertion
1010 en$=cs$+nm$+pr$+ad$+tl$
1020 iadd 1, en$
1030 if ok = 64 then out"FILE FULL",1,1
1040 .....
```

Note : It is important to be sure that the field for the access key is properly positioned inside the record in order to have the right insertion when file is validated. With iadd, two records having the same access key can exist in the same file since MASTER does not check for duplicates. For duplicate management, refer to the paragraph on the ivalidate command. (see 6-18)

iread - indexed file **read** record

Function : **iread** reads a record using the complete primary key (or part of it called a radical).

Syntax : **iread** nf,cl$,en$
nf : file number
cl$: complete key string : strict test
or radical string : generic test
en$: receiving string

Utilisation : The **iread** command reads from the MF file, the record corresponding to the key entered. Once the record is found, it is stored in en$. Because the record reading is executed according to its access key, if the record had been entered into the file by iadd, the file must be validated before the iread command can be executed. Otherwise, the record will not be found. This command enables two kinds of reading : either from a complete key and, in that case, MASTER will retrieve it if it exists, or from an incomplete key (a radical) called a generic retrieval. In the later case, MASTER retrieves the first record the key of which corresponds to the radical. If the radical does not correspond to any record, MASTER sends a message through the ok status variable.

The status variables take the following values :
- ok = 0 : record inside the receiving string
- ok = 1 : record does not exist
- ok = 255 : does not exist, end of file
(test on next key useless)

Example :

```
1000 rem---------------- reading a record
1010 rem----------------strict retrieval
1020 cl$="1234567890123"
1030 iread 1,cl$,en$
1040 if ok = 1 or ok = 255 then out"DOES NOT EXIST",1,1
1050 rem----------------generic retrieval
1060 cl$="1923"
1070 iread 1,cl$,en$
1080 if ok = 1 or ok = 255 then out"DOES NOT EXIST",1,1
1090 ..............
```

iexist - indexed file **existence** of record

Function : The **iexist** command tests the existence inside the MF file, of a record corresponding to the key entered.

Syntax : **iexist** nf,cl$
nf : file number
cl$: complete key string : strict test
radical string : generic test.

Utilisation : As with the iread instruction, it can be a complete key or radical. So the test can be strict or generic (see the previous page). This command only checks for the existence or non existence inside the file. It is a very quick command.

The status variables take the following values :

- ok = 0 : exists
- ok = 1 : does not exist
- ok = 255: does not exist, end of file
(test on next key useless)

Example :

```
1000 rem--------------check
1010 rem strict test
1020 cl$="1234567890123"
1030 iexist 1,cl$
1040 if ok = 1 or ok = 255 then out"EMPLOYEE NOT IN FILE",1,1
1050 rem generic retrieval
1060 cl$="1923"
1070 iexist 1,cl$
1080 if ok = 1 or ok = 255 then out"EMPLOYEE NOT IN FILE",1,1
1090 .....
```

Note : The command only tests for the existence of indexed record that is to say those entered into the file by iwrite and those entered by iadd followed by ivalidate.

idelete - indexed file **delete** record

Function : **idelete** deletes a record from the file using its primary key for identification.

Syntax : **idelete** nf,cl$
nf : file number
cl$: complete key string.

Utilisation : The **idelete** command deletes the record corresponding to the key entered. For this, the key must be complete, otherwise a programming error is detected. If the record to be destroyed does not exist in the file, MASTER transmits a message through the ok variable.

The status variables take the following values :

- ok = 0 : record deleted
- ok = 1 : does not exist
- ok = 255: does not exist, end of the file (test on next key useless).

Example :

```
1000 rem----------------------delete a record
1010 cl$="1234567890123"
1020 idelete 1,cl$
1030 if ok=1 or ok=255 then out "EMPLOYEE NOT IN FILE",1,1
1040......
```

Note : This command only deletes indexed records, that is to say those entered in the MF file by iwrite and those entered by iadd followed by ivalidate.

The space occupied by a deleted record is **not** available to records entered at a later date. For example, writing 1000 records, deleting 500 and writing 500 more will take the place of 1500 records on the disk. However, the records deleted will disappear for **all** intent and purposes, but **copy file** (see D) will reorganize the files.

iupdate - indexed file **update** record

Function : **iupdate** modifies a record already in the file.

Syntax : **iupdate** nf,en$
nf : file number
en$: complete record string

Utilisation : The **iupdate** command modifies a record inside the MF file. The record field defined as the access key cannot be modified. The whole record must be written again, because MASTER overlays the new one on the old one. If the record was entered by iadd the file must first be validated. If the record to be updated does not exist in the file, MASTER transmits an error message through ok.

The status variables take the following values :

- ok = 0 : record updated
- ok = 1 : does not exist
- ok = 255: does not exist, end of the file
(test on next key useless).

Example :

```
1000 rem--------------modifying a record
1010 en$=cl$+nm$+pr$+ad$+tl$
1030 iupdate 1,en$
1040 if ok = 1 or ok = 255 then out"EMPLOYEE NOT IN FILE",1,1
1050 .....
```

Reading in ascending or descending order

Overview

The **istart** and **inext** commands reads **an MF** file in ascending or descending order using the key. To do this, enter the key from where the reading is to start, by using the **istart** command, MASTER retrieves the record corresponding to that key or, if it does not exist, the first record. The **inext** command reads the next record to the one found. The (-) option reads the previous one. Thus, it is possible to move up and down inside the file.

It is possible through MASTER to extract from the file only those records corresponding to a retrieval mask defined in mk$. The selection criteria can be part of the key or part of the record.

When using the **inext** command, in a loop for instance, the other MASTER commands are still available. This permits reading, changing, deletion or entering of a record and, after that, to move directly to the next or previous record.

Utilisation of the retrieval mask

With the retrieval mask, it is possible to work only on the records meeting defined criteria. The mask can be used with the istart (-), inext(-) and idata commands.

The principle is very simple. First define a string composed of ↑ (up arrows) , as long as the record size. It represents a mask without selection. Replace the ↑ with the characters being searched. Thus, a mask like ABCDE↑↑↑↑↑↑↑↑12 takes into account only those records that contain the information : ABCDEF and 12, in those respective positions. An empty string can be used instead of a mask.

- mk$ = "" : no selection

- mk$ = "↑↑↑↑↑75" : only those containing 75 as the sixth and seventh characters.

istart - indexed file **start** getting records

Function : istart is used to initialize a reading in ascending or descending order using the key.

Syntax : istart [-] nf,cl$,mk$,en$
without optional symbol: ascending order
with optional symbol : descending order
nf : file number
cl$: search key
mk$: retrieval mask (see 6-15)
en$: receiving string

Utilisation : Send a complete or generic key in a mask. MASTER will search for the record corresponding to the mask definition or the closest one. It is possible to read the records in ascending or descending order, by adding the (-) option to get the previous or the following record if the record wanted does not exist.

SUMMARY :

1) Place the (-) symbol before the file number for descending reading,
2) mk$ = "": no selection,
3) add"↑" or chr$(94) to mk$ for an extraction,
4) the first key can be a complete or a generic one,
5) cl$="": positions at the beginning of the file. chr$(0) where key is packed
6) cl$ = chr$ (250) : positions at the end of the file.

Status variable values :
- ok = 0 : Retrieves the record corresponding to the key
- ok = 1 : No record for this key. Retrieves the first record
- ok = 255: No record and end of the file (according as -/+).

Example :

```
1700 rem start reading records
1710 rem from key begining by "1234"
1720 cl$="1234":mk$=""
1730 istart 1,cl$,mk$,en$ : rem -1 : previous record
1740 if ok = 255 then out"NO RECORD",1,1
1750 goto.....loop inext
```

inext - indexed file get **next** record

Function : inext is used, after istart, for the search in order by key.

Syntax : inext [-] nf,mk$,en$

without optional symbol : ascending order
with optional symbol : descending order
nf : file number
mk$: retrieval mask (see 6-15)
en$: receiving string

Utilisation : Once the istart command is executed, the inext (-) command can be used to move up or down in the file, in the key order. Sending a retrieval mask and a target string, MASTER retrieves the record that follows the last one accessed by **inext** or istart which corresponds to the mask defined. If the - was typed it will retrieve the previous one. The istart command does not have to be used to reverse the reading order. The mask can also be modified.

1) type - for a descending reading
2) mk$: same use as with istart
3) iread, like istart, initializes the file (but the complete or generic key must exist inside the file).

Status variable values :

- ok = 0: record retrieved into the receiving string
- ok = 1: record retrieved into the receiving string
  The difference between ok=0 and ok=1 lies in the fact that with the latter it is possible to insert a record between the one displayed and the previous one.
- ok = 255: end of file according as the - option.

Example :

```
1800 rem-------------reading loop
1810 inext 1,mk$,en$ : rem -1 previous record
1820 if ok=255 then out "END OF FILE",1,1
1830 out en$,1,1
1840 goto 1810
```

ivalidate - indexed file **validate**

Function : ivalidate updates the index table with all the records entered by iadd (batch)

Syntax : ivalidate nf [,n]
nf : file number
n : (optional) zone number for display
(see utilisation)

Utilisation : MASTER adds to the index all the records entered by iadd since the last validation, so that nr=ni (except in case of duplicates).

The validation can be done at any time.

When a record is entered by iadd, MASTER does not check for duplicates, if there are duplicates, MASTER will only enter the first record into the index. ok = 1, indicates there are duplicates.

In that case, ni is smaller than nr, the difference being the number of duplicates.

Executing this command might take some time. To keep tract of what is happening add to the ivalidate command, an output data zone (defined with the decz command : see MASTER SCREEN) into which after each record validation, MASTER will display nr-ni. This zone can be formatted.

The values of the status variable are :
- ok = 0 : execution ok
- ok = 1 : duplicates in the file
- ok = 255: all the keys are inserted

Example :

```
1900 rem---------------------- validate
1910 ivalidate 1
1920 if ok= 1 then out "DUPLICATE KEY",1,1
```

irestore - indexed file **restore** pointers
idata - indexed file read **data**

Function : The use of both instructions enables a sequential reading of the file.

Syntax : **irestore** nf
idata nf,mk$,en$
nf : file number
mk$: retrieval mask
en$: receiving string

Utilisation : The **idata** command reads a file in the order the records were entered. The **irestore** command sets pointers to the beginning of the file in order to initialize the reading. Using the retrieval mask it is possible to skim quickly through the file. **idata** reads all the records, including those entered by iadd and not yet validated. The use of the mask is similar to inext.

The status variables take the following values :

- ok = 0 : record in the receiving string
- ok = 255: end of file

Example :

```
2000 rem----------------reading in order of creation
2010 irestore 1
2020 idata 1,"",en$
2030 if ok=255 then 2060: rem end of file
2040 out en$,5,1
2050 goto 2020
```

iclose - indexed file **close**

Function : **iclose** closes the files.

Syntax : **iclose** nf
nf : file number

Utilisation : The **iclose** command closes an MF file. This means that, it saves the index related to a file. This is the reason why it is imperative to use it after the following commands :

- iwrite
- idelete
- ivalidate

Trying to close a file not opened gives a "file not open" message.

iclose frees the space reserved for the index in memory. This space can be used again by MASTER or BASIC.

Example :
```
2060 rem----------end of file utilisation
2070 iclose 1
2080 end
```

.....

iupgrade - indexed file **upgrade**

Function : **iupgrade** saves the index table on disks.

Syntax : **iupgrade** nf
nf : file number

Utilisation : The **iupgrade** command is iclose followed by iopen. It is to be used if you want to keep the index opened after the processing is finished. Saving the index protects the file. Trying to **iupgrade** a file not opened gives a "file not opened" message.

Example :
```
2060 rem----------backup index table
2070 iupgrade 1
2080 rem----------next step in program
```

.....

Note : **dclose cannot be used before iclose or iupgrade;** this results in a "NO CHANNEL" error. The dclose may be used after an **iclose** or **iupgrade** command.

ireset - indexed file **reset**

Function : ireset initializes MASTER

Syntax : ireset

Utilisation : This command resets MASTER. All MASTER files **must** be closed (iclose) before activating **ireset**, otherwise the indexes will be destroyed.

The command also resets the screen pages stored in memory. MASTER returns to BASIC all memory used for files, tables, screen and print page management with an **ireset.**

Example : 5 ireset

Note : This instruction must be used with care. Be sure to save all information on disk before it is used.

MASTER FILE
STATUS VARIABLES

SUMMARY

These five variables are reserved for MASTER. They are used as interfaces between MASTER and the application or the programmer (like zo for the screen generator).

nr : This variable indicates the number of records in the file. Incremented by iwrite and iadd.

ni : Number of records in the index tables. Incremented by iwrite and ivalidate decreased by idelete.

lr : Record length of the file (evaluated after iopen).

lk : Key length of the file (evaluated after iopen).

ok : Status variable

```
ok = 0   : everything ok
ok = 1   : isam error (according to the command)
ok = 64  : file full
ok = 128: file not found
ok = 255: beginning or end of file
```

Caution : ok only controls MASTER status like existence of a record in a file. The ds and the ds$ variables can be handled as usual. The error command is released when ds is different from 0, and not with the ok value; ds value is stored in st.

Note : These variables take their status when accessing the file. In case of a multifile, it will be for the last one queried (except for lr and lk holding the last iopen values).

Physical organization of the MF files on disk

An MF file is registered in the disk catalog as a file of USR type. For one MF file, there will be three USR files in the catalog.

The number of blocks allocated for these files, registered in the catalog, is fixed and insignificant. On the contrary, the number of free blocks is significant.

- The file, the name which ends with a "d", contains all the records, "stacked" according to the chronological order of creation.

- The file, the name of which ends with an "I" contains the access keys to the records, arranged in a tree-like structure.

- The third file contains the table to be loaded into memory with "iopen" and saved on a disk with "iclose". This table represents the first level in the tree-like structure.

According to the option choosen (see RESERVE FILE), the access key file will consist of one or two levels.

- The links between keys and records are created in a physical way (that is to say track and sector). Therefore, **the collect instruction will damage an MF file.** Likewise, to copy an MF file, **do not** use the copy instruction of the BASIC. Use the COPYFILE utility included in MASTER (see D).

An access key to a record is included twice in an MF file :

- in the record itself ("d" file),
- in the index ("I" file)

An MF file is called an "internal key" file. This redundancy is necessary for security reasons. The fact that all the data is in the "d" file, means in case of total or partial destruction of the index or of the table in memory, or in case of an accidental destruction of the links (for instance, misusing the collect instruction), the "d" file will rebuild the whole file (see the appendix).

Utilisation of the disk access channels by MASTER

To handle the files, MASTER uses only one random access channel.

To avoid interferences between MASTER and the DOS (access to a sequential or relative file), remember that :

- MASTER file's logical number is 126. This number is so reserved for MASTER.

- The random access file is opened with the first file opening ; it is closed with the last file closing.

- The inblock and putblock commands (See 7-6, 7-7) open temporarily the channel if necessary.

- MASTER chooses the floppy channel number; it takes the first free one from number two.

- The dopen BASIC command that chooses automatically a free channel can be used with MASTER, no precautions are necessary (besides dopen > 125 which are reserved for MASTER).

- Since the "open" command chooses the floppy channel number, be careful not to choose the same one as MASTER. The use of this command is not recommended. It is better to use "dopen".

When using the print generator, another access channel is opened on a relative file. The logical number is 125 and the disk access channel is chosen in the same way as with the random access channel. (The first free one from the second channel).

Also take into account that with the CBM disk (1541 or other), the number of channels opened at the same time is limited to 5, a relative file (similar to MASTER PRINT) takes two.

Overview

The packing commands were designed to optimize disk space. It is understood that, for MASTER, a record is a string of a fixed length and that a field of this string is used as an access key. There are 4 types of data, that can be mixed.

- alphanumeric type : c
- binary type : s
- unsigned integer type : b
- floating point type : f

and a fifth separate one

- padded type : p which is a particular kind of alphanumeric type

Each must be packed in a different way. A precise calculation of the packed length of each field is necessary to know the total length of the packed record for the use of RESERVE FILE.

Packing is performed by the **PUTST** instruction
Unpacking is performed by the **GETST** instruction

The **CREATST** function creates strings of fixed length with fixed characters and will be used frequently in the packing operation.

Organization of the tasks for the definition of a packed file

1 - File definition

- Definition of the record and its fields
- Definition of each field (length and type)
- Calculation of the length of each packed field
- Calculation of the total length of a packed record.

2 - File utilisation

- iopen — open file
- putst en$... — packing of a record

  writing of the record
  and
  reading of the record

- getst en$... — unpacking of a record
- iclose — close file

Example

Each type can be packed to save disk space. The packing operation is different for each type. For example a record containing :

- a$: alphanumeric string of 15 characters,
- b$: numeric string of 13 digits,
- c : unsigned integer,
- d : floating point number.

The en$ record is composed as follows :

en$ =a$+b$+str$(c)+str$(d), which gives a length of :

15 + 13 + 5 + 11 = 44 characters.

Now to pack all the fields of the record.

First calculate each field length once it is packed.

Note

It is important that a field is always the same length. For example a$, string variable, used for a first name "Frank" is 5 characters long ; "John" is 4 characters long. It is imperative that the length of a$ stay equal to 15, otherwise the packing operation will not be performed correctly as other characters will be included with the first characters of the record. Ten blanks must be added to Frank and eleven blanks to John.

To do this use CREATST(see 6-31), also the padded type -p can be used to increase the string.

Alphanumeric packing - c type

This type of packing encodes an alphanumeric character (ASCII CODE between 32 and 96) in 6 bits. Capital letters are returned as small letters.

Calculation of Length

n : length of the string to be packed
lc : length of the packed string
lc = n - int (n/4)

Example

a$ = 15 characters therefore n = 15
lc = 15 - int (15/4) = 12

Note

A TABLE WITH ALL THE PACKED VALUES OF C TYPE IS ON PAGE 6-33

Binary packing - s type

Is used to pack numbers stored in strings (for instance, from the multiprecision arithmetic). Only the integer part is packed. So mmul can be used to get a string ready for packing. A good example of s type is a social security number.

Calculation of Length

n : number to be packed (not its length).

One byte can store a number equal to $2^7 - 1$ (7 and not 8 because the first bit = sign)

Two bytes can store a number equal to $(2^7 * 2^8) - 1$

Three bytes can store a number equal to $(2^7 * 2^8 * 2^8) - 1$

and so on

So, knowing n, find the smallest value corresponding to a maximum number depending on the number of bytes.

Note

A TABLE WITH ALL THE PACKED VALUES OF S TYPE IS ON PAGE 6-34

Unsigned integer packing - f type

This type of packing is used for integers smaller than 32767

1 < n < 128 lc = 1 byte
127< n < 32767 lc = 2 bytes

Floating point number packing - b type

Whatever that number is, it will be packed in 5 bytes

A number lc = 5 bytes

Padded - p type

A string in a packed record (and in a file in general) must have a constant length.

The padded type - p inserts a string into a larger string, of a defined length, adding blanks if necessary.

Example

```
XXXXXXXXXXXXXXXXXXXX
12345678901234567890
0         1         2
```

A string of 20 "X".
A string with the name John : 4 characters. The length must be ten characters and start at the fifth position. The padded - p type will do the job (exact syntax for GETST and PUTST ,6-32,6-33)

```
XXXXJOHN      XXXXXX
12345678901234567890
0         1         2
```

The padded type is not a packing type, but it will simplify the packing operation in the creation of the complete string. Also, the choice between the alphanumeric type (c) and the padded type (p) depends on the relation speed/space on disk. If you have room enough, the padded type will speed up your packing/unpacking operations. If space is at a premium the c - type should be used.

It is possible to have both types without problems. The choice is up to the programmer.

Note Once packed, the en$ string changes from 44 characters to : 12 +7 +2 +5 = 26 characters, which represents about 40% saving on the disk.

creatst - create a string

Function : creatst creates a string

Syntax : creatst a$, ln [,chr$ (ca)]
ca : value of an ASCII character
ln : wanted length of the string
a$: name of the string

Utilisation : The **creatst** command creates a string of chr$(ca) ln characters long. In the absence of chr$ (ca), chr$ (32) will be used. For packing any character can be used. This instruction can be used to create retrieval masks.

Example : creatst a$, 80

will create a string a$ of 80 blanks

creatst b$,10,"*"

will create a string b$ with 10 stars.

Note : This instruction replaces the following BASIC loop

```
for i = 1  to ln
a$ = a$ + chr$ (ca)
next i
```

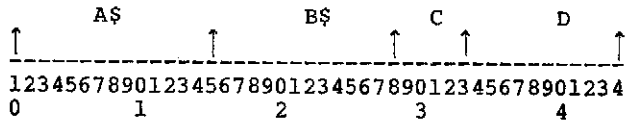
putst - **put** in **st**ring

Function : putst compacts a record (a collection of fields)

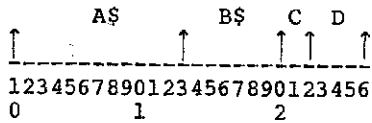
Syntax : putst a$;ty,b$,pt,ln[;][...
a$: receiving string,
ty : type (p,c,s,b,f),
b$: variable to pack,
pt : pointer inside the receiving string,
ln : string length once it is packed.

Utilisation : putst is used each time a record has to be packed (before it is stored on disk), pt and ln are better explained in the example.

Example :



Original record **total : 44 characters**



Packed record **total : 26 characters**

```
creatst en$,26
putst en$;c,a$,1,12;s,b$,13,7;b,c,20,2;f,d,22,5
```

Note : Error messages
"out of string" : receiving string too short
"out of format" : the packed string length is greater than the ln length.

getst - get from **string**

Function : getst unpacks a string (a record or a collection of fields)

Syntax : getst en$;ty,a$,pt,ln [;][;
en$: string to unpack
ty : type (p,c,s,b,f,)
a$: receiving variable (string or num)
pt : pointer inside the packed string
ln : packing length of the data

Utilisation : Once a packed record is read from disk, **getst**, using the same parameters as for putst, unpacks the record and dispatches the unpacked values into the corresponding variables.

Example : (See 6-30)
getst en$;c,a$,1,12;S,b$,13,7;b,c,20,2;f,f,22,5

Note : 1) The unpacking of a non-packed string (or record) must be avoided.

2) Since the packed strings of a type c are returned in small letters, you must use the **uplow** instruction to get them in capital letters. (see uplow 7-5)

Important : Packing of the primary key

The first byte of the key, if it is to be packed, must be smaller than 250.

This limitation creates a few problems.

- The c type (alphanumeric) is not permitted.
- The f type (floating point) is not permitted.
- The b type (unsigned integer) is permitted provided the number is smaller than 32767.
- The s type (binary) is permitted provided the number is greater than 0.
- The p type (padded) is permitted.

Note : The p type is best used for the primary key in all circumstances. However the s type (binary) may be used, for example with a Social Security Number.

Alphanumeric Packing C - Type, Packing Value Table
N - Orig. String Length / lc - Packed Length

N	lc	N	lc	N	lc	N	lc	N	lc
1	1	52	39	103	78	154	116	205	154
2	2	53	40	104	78	155	117	206	155
3	3	54	41	105	79	156	117	207	156
4	3	55	42	106	80	157	118	208	156
5	4	56	42	107	81	158	119	209	157
6	5	57	43	108	81	159	120	210	158
7	6	58	44	109	82	160	120	211	159
8	6	59	45	110	83	161	121	212	159
9	7	60	45	111	84	162	122	213	160
10	8	61	46	112	84	163	123	214	161
11	9	62	47	113	85	164	123	215	162
12	9	63	48	114	86	165	124	216	162
13	10	64	48	115	87	166	125	217	163
14	11	65	49	116	87	167	126	218	164
15	12	66	50	117	88	168	126	219	165
16	12	67	51	118	89	169	127	220	165
17	13	68	51	119	90	170	128	221	166
18	14	69	52	120	90	171	129	222	167
19	15	70	53	121	91	172	129	223	168
20	15	71	54	122	92	173	130	224	168
21	16	72	54	123	93	174	131	225	169
22	17	73	55	124	93	175	132	226	170
23	18	74	56	125	94	176	132	227	171
24	18	75	57	126	95	177	133	228	171
25	19	76	57	127	96	178	134	229	172
26	20	77	58	128	96	179	135	230	173
27	21	78	59	129	97	180	135	231	174
28	21	79	60	130	98	181	136	232	174
29	22	80	60	131	99	182	137	233	175
30	23	81	61	132	99	183	138	234	176
31	24	82	62	133	100	184	138	235	177
32	24	83	63	134	101	185	139	236	177
33	25	84	63	135	102	186	140	237	178
34	26	85	64	136	102	187	141	238	179
35	27	86	65	137	103	188	141	239	180
36	27	87	66	138	104	189	142	240	180
37	28	88	66	139	105	190	143	241	181
38	29	8	67	140	105	191	144	242	182
39	30	90	68	141	106	192	144	243	183
40	30	91	69	142	107	193	145	244	183
41	31	92	69	143	108	194	146	245	184
42	32	93	70	144	108	195	147	246	185
43	33	94	71	145	109	196	147	247	186
44	33	95	72	146	110	197	148	248	186
45	34	96	72	147	111	198	149	249	187
46	35	97	73	148	111	199	150	250	188
47	36	98	74	149	112	200	150	251	189
48	36	99	75	150	113	201	151	252	189
49	37	100	75	151	114	202	152	253	190
50	38	101	76	152	114	203	153	254	191
51	39	102	77	153	115	204	153	255	192

Binary packing s type, Packing Value Table.
Nmax - Maximum N Value for a Packed Length of lc

Nmax	lc
127	1
32767	2
8388607	3
2.14748365 e+9	4
5.49755814 e+11	5
1.40737488 e+14	6
3.6028797 e+16	7
9.2233720 e+18	8
2.36118324 e+21	9
6.0446291 e+23	10
1.54742505 e+26	11
3.96140813 e+28	12
1.01412148 e+31	13
2.59614843 e+33	14
6.64613998 e+35	15

Overview

Grouped under the name MASTER BASIC are all the functions which do not depend directly on the three generators (M. SCREEN, M. FILE, M. PRINT). These instructions simplify the controls, improve some performances of BASIC and ease the programming.

These functions include :

These functions are all independant and can be studied separately.

madd - multiprecision **addition**

Function : Addition with multiprecision.

Syntax : **madd** n1$,n2$,n3$

n1$,n2$: string of the numbers to be added
n3$: result string

Utilisation : To be used for operations needing more than 8-digit precision. n1$+n2$=n3$

Example :
```
n1$="1111111111"
n2$="2222222222"
madd n1$,n2$,n3$
out n3$,10,1
3333333333.00000000000
```

Note : (see note mdiv 7-3)

.....

msub - multiprecision **subtraction**

Function : Subtraction with multiprecision.

Syntax : **mmsub** n1$,n2$,n3$

n1$ first number
n2$ number to be subtracted
n3$ result

Utilisation : (see madd) will perform n1$-n2$ and store the result in n3$.

Example :
```
n1$="333333333333"
n2$="111111111111"
msub n1$,n2$,n3$
out n3$,10,1
222222222222.000000000
```

Note : (see note mdiv 7-3)

mmul - multiprecision **mul**tiplication

Function : Multiplication with multiprecision

Syntax : **mmul** n1$,n2$,n3$

n1$,n2$: numbers to multiply
n3$: result

Utilisation : Multiply the two numbers n1$ ad n2$ and store the result in n3$
n1$xn2$=n3$

Example : n1$="11"
n2$="22"
mmul n1$,n2$,n3$
out n3$,10,1
242.0000000000000000000

Note : (see below)

.....

mdiv - multiprecision **div**ision

Function : Division with multiprecision

Syntax : **mdiv** n1$,n2$,n3$

Utilisation : Divide the content of n1$ by n2$ and store the result in n3$

Example : **mdiv** "1","3",b$
out b$,10,1
.33333333333333333330

Notes : The **mdiv** and **mmul** instructions do not accept empty string or the null value. The multiprecision commands calculate the four operations with a precision of 22 digits and one exponent from + 63 to - 64. Results are displayed in scientific notation if the exponent is above 22 and below -3.

goto

Function : **goto** lb branches on a program line given by a variable.

Syntax : **goto** lb
lb : variable containing a line number.

Utilisation : It stores the line number in a variable, now able to branch the program using that variable.

Example :
```
10 input "30 or 40";a
20 goto a
30 print "30":stop
40 print "40":stop
```

---

gosub

Function : **gosub** lb branches on a subroutine, the address being given by the variable lb.

Syntax : **gosub** lb
lb : variable containing a line number.

Utilisation : Similar to goto.

Example :
```
10 input "30 or 40";a
20 gosub a
30 print "30":stop
40 print "40":stop
```

uplow - uppper case / lower case conversion

Function : **uplow** inverses all the letters of a string.

Syntax : **uplow** a$
a$: any string

Utilisation : In a string of characters, all the upper case is changed to lower case and vice versa.

Example : a$="PATRICIA"
uplow a$
out a$,10,10
patricia

Note : **uplow** can be used after an unpacking operation of c-type because an unpacked string is always in lower case.

putbuf - put in **buffer** (from string)
putblock - put in disk **block** (from buffer)

Function : **putbuf** transfers a string to a buffer.
putblock transfers the buffer to disk.

Syntax : **putbuf** a$,pn
putblock tr,se[,nd][,nu]

a$: string to be written on disk
pn : pointer where a$ must go in the buffer
tr : track number - disk
se : sector number - disk
nd : drive number (0 if not defined)
nu : disk logic number (8 if not defined)

Utilisation : These commands write in random access mode, without using BASIC rather tedious syntax. Opening a channel is not necessary to perform these commands.

Example : a$="DATA"
putbuf a$,1
putblock 10,10,0 : rem we must have a disk in the drive

Note : The **putbuf** and **putblock** instructions have nothing to do with MASTER FILE and are not to be used with that file management system.

inblock - in buffer from disk **block**
takbuf - take from **buf**fer to string

Function : **inblock** transfers a disk block to a buffer.
takbuf transfers a buffer in a variable.

Syntax : **inblock** tr,se[,nd][,nu]
takbuf a$,pn,ln

tr : track number - disk
se : sector number - disk
nd : drive number (0 if not defined)
nu : disk logic number (8 if not defined)
a$: receiving string
pn : pointer from which the data must be transfered in the string
ln : number of characters to put in the string.

Utilisation : These instructions facilitate a random access reading. Opening a channel is not necessary.

Example : inblock 10,10,1
takbuf b$,1,6
out b$,10,10
DATA

Note : (see 7-6)

nolist - no listing save

Function : saves a program that can never be listed again.

Syntax : nolist "name"[,dr]
name : program name (< 16 characters)
dr : drive number (d0 if not defined)
(d0 or d1)

Utilisation : The **nolist** command, like dsave, saves a program, but the program saved cannot be listed again.

Trying to list a program saved by **nolist**, generates a reset of the computer.

Example : nolist "prg1"

Notes : The **nolist** command will work with the MASTER 64 development version, but not with the run-time version on the diskette. A protected run-time version using electronic key is available that accepts **nolist** programs. (See 3-2 and Appendix G)

OVERVIEW

MASTER BASIC controls dates (leap years included). Send the reference date through the reserved variable: dr$, in the form of dd/mm/yy with the required separators. The date to control is stored in the da$ variable with the separators.

These, dr$ and da$, are 2 reserved variables for MASTER and cannot be used when datepack and dateunpack are used.

The **datepack** command operates if da$ is valid. If there is an error, ok = 10. Once the date is controlled, it is stored in da in two bytes.

The **dateunpack** command unpacks the date back to the da$ date with the separators defined in dr$. The date to unpack must be stored in da.

Therefore, the **datepack** and **dateunpack** commands need no parameters.

DA STRUCTURE (A.P.)

da is calculated as follows
dr$=dr/mr/yr
da$=da/ma/ya
d1= (12x(ya-yr))+(ma-mr) represents the number of months
d2= da-dr
da= d1x256+d2

If packed in two bytes (as for instance : putsta$;b,da,1,2), on the first byte, the number of months that have passed since the reference date (+12 months per year) and, on the second one, the number of days since the reference date (the difference can be negative.)

Note :
dr$ is the reference date. The control on da$ (also the packing) will not be done if dr$ is more than 10 years prior to da$. Also da$ can not be earlier than dr$.

datepack - **date packing**
dateunpack - **date unpacking**

Function : **datepack** packs a controlled date.
dateunpack unpacks a controlled date.

Syntax : **datepack**
No parameters but the date to be packed must be in da$ and the reference date must be in dr$

da$ / dr$ = "dd/mm/yy"
dd - day
mm - month
yy - year

dateunpack (the packed date held in da.)

Utilisation : dr$ must have a value before **datepack** can be used. If the date is correct, the operation will be validated and the date stored in two bytes (da).

With an already packed and controlled date held in da, **dateunpack** will unpack the date and store it in da$.

Example :
```
dr$="01/01/81"
da$="07/04/82"
datepack
print ok
0

da$=" "          (previous example continued)
dateunpack
print da$
07/04/82
```

Note : 1) - If a bad date is found, the packing is not done and the status variable ok = 10.

2) - Other characters than "/" can be used to seperate day, month, and year in da$ and dr$. But only one character can be used at a time

Example : dr$="01-01-83":da$="17-08-83"

hunt

Function : **hunt** will find the position of a character within a string.

Syntax : **hunt** [-] chr$(ca),a$,pn

- : optional symbol
  with [-] : search for the first character equal to,
  without [-]: search for the first character not equal to
ca : decimal value of an ASCII character
a$: string where the search is to be done
pn : pointer within the string to indicate where the search starts.

Utilisation : **hunt** allows control of such operations as acquisition, validation, and also statistics. The result will be found in zo.

Example :
```
a$ = "patricia owen"
hunt chr$(65),a$,1:rem 65 : a
? zo
2

hunt - "e",a$,1
?zo
1
```

Note : If the character does not exist, zo=0

Overview

EDEX is a programming aid tool entirely written in 6502 machine language

It simplifies a lot of the programming and debugging with 15 extra instructions.

- **Listing instructions,**
  (auto, delete, renu)
- **Debugging instructions,**
  (dump, error, find)
- **Printing instructions,**
  (hardcopy, plot, reset, pusing)
- an **if then else** instruction,
- a **beep** instruction,
- a **trace** utility,
- and **various functions** including an inhibition of the STOP key.

Note : Most of the instructions can be used in program mode or direct mode.

auto - **auto**number program lines

Function : **auto** automatically numbers the lines of a program.

Syntax : **auto** n
n : increment (1 <= n <= 255)

Utilisation : To enter the auto line numbering, type **auto** and **return**. To start, type the first line of the program. After typing **return** on that first line, the next line number will appear.

It is still possible to enter any number to change the automatic line number.

To stop the auto line numbering, enter the command **auto** without any argument. It will also stop by typing **return** after line number.

Example : **auto** 10
10 for i = 1 to 10
20

Note : **auto** can be used in direct mode only.

delete - **delete** program lines

Function : **delete** deletes lines of a program.

Syntax : **delete** a
delete a-
delete -a
delete a-b

a,b : line numbers

Utilisation : **delete** deletes lines with the same syntax as the LIST function

1) **delete** a deletes line a.

2) **delete** a- deletes all lines between line a and the end of the program.

3) **delete** -a deletes all lines from the beginning to line a.

4) **delete** a-b deletes all lines between line a and line b.

Example : **delete** 20

Notes : - **delete** without argument is not valid.
- **delete** can be used in direct mode and program mode.

renu - **renu**mber program lines

Function : renu renumbers the lines of a program.

Syntax : renu [a[,b[,c]]]
all 3 arguments optional
(see utilisation).

Utilisation : renu will renumber all lines of a program (including GOTO, GOSUB, etc...)

Four possibilities :

1) **renu**
Renumbers the whole program
The first line becomes 100
The line increment is 10

2) **renu** a
Renumbers the whole program
The first line becomes 100
The line increment is a

3) **renu** a,b
Renumbers the whole program
The first line becomes a
The line increment is b

4) **renu** a,b,c,
Renumbers a part of the program starting at line a
That line a becomes b
The line increment is c

Example : renu 20,10
This will renumber a whole program, the first line becoming 20, with a line increment of 10.

Note : only in direct mode

dump - **dump** variables

Function : dump gives the list of the variables used in a program and their values.

Syntax : dump

Utilisation : dump can be used at any time
- after a run of a program,
- after a STOP (and continue : CONT),
- after a syntax error.

dump does not give the content of arrays to avoid an overload of the screen.

Example : a = 2 : b = 3 : a$ = "test"
dump

will give
a = 2
b = 3
a$= "test"

Note : Can be used in program and direct mode.

error - **error** in BASIC line

Function : error gives the location of an error in a BASIC line.

Syntax : error with no argument

Utilisation : When an error occurs in a program, BASIC stops the execution and gives an error message. To locate the error, just type

error

error will display the faulty line and the error will be highlighted in reverse video.

Example :

```
10 a$ = STR$(a
20 end
run
error
```

Note : only in direct mode.

No other commands must be used before **error** when the interruption occurs. (not even LIST).

find - **find** character in string

Function : **find** finds any string of characters within a program.

Syntax : **find** <de> ch <de> [A - B]

<de> : almost any character to be used as a delimiter

ch :characters being sought

A-B: (optional)
A : line number where the search starts
B : line number where the search stops.

Utilisation : To find any character or string of characters. This can be very useful when correcting syntax errors. Just search for the name misspelt etc..

Example :

```
10 print "write"
20 print "red"
30 print "blue"
40 print "glu"
50 end

find "glu"

find @end@
```

Note : can be used in program and direct mode.

hardcopy

Function : **hardcopy** prints an exact copy of the screen on the printer.

Syntax : **hardcopy** without argument in program mode

Utilisation : as simple as it looks.

Example : 10 hardcopy
run

Notes

The use of this function will stop the internal clock of the C-64 during its execution.

Can be used in direct or program mode.

plot - **plot** point on screen

Function : plot draws on the screen. The definition is 80 x 50 (4000 points)

Syntax : plot a,b
a : x coordinate (from 0 to 79)
b : y coordinate (from 0 to 49)
(0,0) : starting point : lower left-hand corner of the screen.

Utilisation : look at the example program on next page.

Example :
```
5 sclear
10 for i=0 to 2*3.14 step.04
20 plot 40 +35*sin(2*i),25+20*sin(3*i)
30 next
40 end
```

---

reset - **reset** point on screen

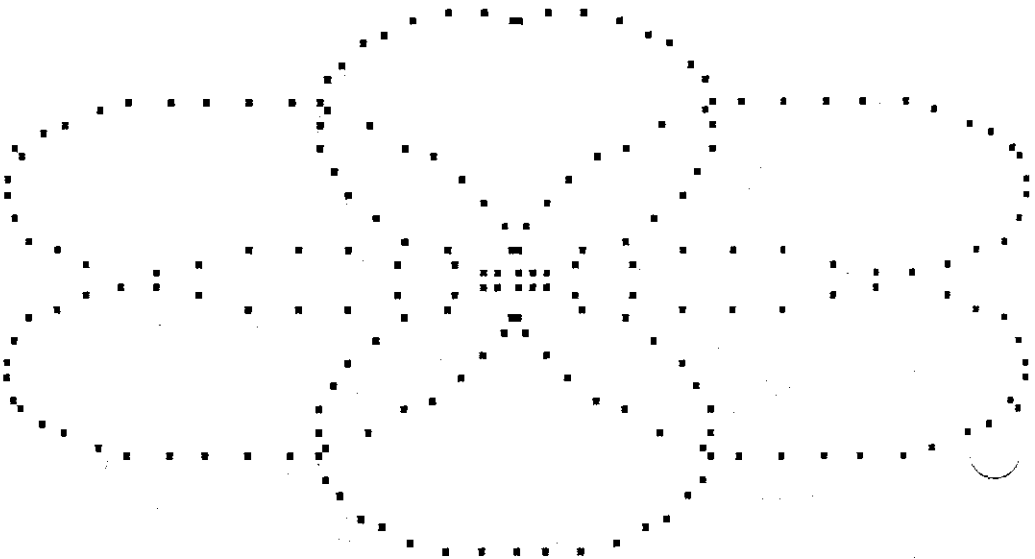
Function : reset erases results of plot routine.

Syntax : reset a,b
a : x - coordinate
b : y - coordinate.

Utilisation : reset clears all drawing done by plot.

Example : (see next page)

Note : plot and **reset** can be used in program and direct mode.



```
10 pi = 3.14159265
20 sclear
30 for i = 0 to pi*2 step pi/100
40 x = 39 + 39*sin(2*i)*cos(3*i)
50 y = 24 + 24*cos(2*i)*cos(3*i)
60 plot x,y
70 next i
80 for i = 0 to pi*2 step pi/100
90 x = 39 + 39*sin(2*i)*cos(3*i)
100 y = 24 + 24*cos(2*i)*cos(3*i)
110 reset x,y
120 next i
```

pusing - print using an edition format

Function : **pusing** prints, on the screen or the printer, in a formatted way.

Syntax : **pusing** f$,a$,b$,...,a,b..
pusing#x,f$,a$,b$,..,a,b...
f$: format
a$,b$,a,b : variables
x : peripheral file number

Utilisation : The format is defined in a string using control characters and **pusing** will do the editing. This can sometimes replace MASTER SCREEN functions.

Control characters :

A - alphanumeric
9 - numeric with blank when no value before decimal point
Z - numeric with zero when no value before decimal point
. - decimal point

, - separation of the different fields of the format.

Example : f$ = "zzz.99,99999"
pusing f$,129,345

Note : Can be used in direct or program mode.

beep

Function : **beep** produces a sound.

Syntax : **beep** with no argument.

Utilisation : This instruction is useful when displaying messages to the screen, or if data must be entered.

Example : out "this is a beep!",1,1:**beep**

Note : can be used in program and direct mode.

if then else

Function : **if then else** is an improved version of the if then allowing for better programming.

Syntax : 10 if<condition>**then**<instruction 1>:**else** <instruction 2>
OR
10 **if** <condition> **then** <instruction 1>
20 **else** <instruction 2>

Utilisation : If a condition is true, then instruction 1 will be executed. If the condition is untrue, then the instruction 2 will be executed. Up to 16 nested **if-then-else** are possible.

Example :
```
10 if <c1> then <i1>
20   if <c2> then <i2>
30     if <c3> then <i3>
40     else <e3>
50   else <e2>
60 else <e1>
```

c1 c2 c3 : conditions or tests and
i1,i2,i3,e1,e2,e3: instructions

Example :
```
10 a = 2
20 a = a + 1
30 if a = 2 then ? "a=2" : else ? "a=3"
40 end
```

run this program twice; the first time with the line 20, the second without.

Note :
- can be used only in program mode.
- up to 16 nested **if-then-elses** and the number of **elses** must be equal or smaller than **ifs**.

trace - **trace** thru program
off - **off** trace mode

Function : **trace** executes a program one line at a time.
off stops the trace function.

Syntax : **trace** with no argument
off with no argument

Utilisation : When the **trace** mode is used, the program runs one line at a time. The executed line is displayed on the screen.

- to display the line, press the **CTRL or COMMODORE** key on each instruction.

- to go to the next instruction release that same key.

- to go faster, press a **SHIFT** key.

- to stop a program which is in **trace** mode, press both **stop** and **release** keys. Then, type **off**.

Example : In program mode, if an error is encountered in subroutine 1000 **trace** through it :

```
100 rem your program
110 trace
120 gosub 1000
130 off
140 rem the rest of your program.
```

Note : can be used in program mode and direct mode.

STOP disabling

With EDEX, it is possible to disable the **STOP** key, so, preventing accidental interruption of the program.

To disable the **STOP** key : poke 1001,1

To enable the **STOP** key : poke 1001,0

BASIC 4.0 COMMANDS

Overview

COMMODORE computers use BASIC as built-in programming language. With the evolution of the machines came modifications of the language and up to now, they are four official versions numbered 1.0, 2.0, 3.0 and 4.0. The C-64 uses the 2.0 version. To be compatible with the 4000 and 8000 series which are using the BASIC 4.0, MASTER includes that BASIC and adds 16 new instructions to BASIC 2.0. to deal with the disk peripheral.

List of Instructions

Housekeeping Operations

These instructions help the user in dealing with the files for operations like copy, delete, rename, etc.. They are used very often and mainly direct mode.

File Handling Operations

These instructions allows to work with two types of files : sequential and relative. Relative files do not exist on BASIC 2.0 and they are very handy.

Disk Status

These two BASIC variables give valuable information on the state of the disk. They help manage any problem occuring to a file operation in direct or programmed mode.

BASIC 4.0 COMMANDS

BASIC 2.0 was created when the main storage peripheral was the tape-drive unit. For example, the command

```
load "prog. name"
```

without any more parameters loads a program from the tape-drive. If you wanted to load from the disk drive, you would have to precise a peripheral number (8).

```
load "prog. name",8
```

BASIC 4.0 introduces new commands to allow to work with the disk drive with more ease. The loading of a program becomes :

```
dload "prog. name"
```

In general, BASIC 4.0 makes the use of a disk drive much more easy as BASIC 2.0 was not created to deal with disks and has a pretty awckward syntax.

Notes :

1) - All BASIC 2.0 commands are still available.

2) - General Syntax :

BASIC 4.0 is designed to be used with more than a single drive. It can be used with more than a unit. So the syntax of most of the instructions include a drive number (d0 or d1) and a disk unit number (u8 or u9). If you are using a 1541 disk drive, you don't need those parameters. So, we will present the instructions in their simplified version, as used with a single drive. The complete syntax will be described in the notes.

- d(n) : drive number - Ex. : d0 or d1
- u(n') : unit number - Ex. : u8, u9

3) - Default values : Drive 0 - Unit 8

4) - When a variable or an evaluated expression is used as a file or program name, it must be surrounded by parentheses.

```
10 dload "program"              is equivalent to:
10 a$="program":dload (a$)
```

dload - disk **load**
dsave - disk **save**

Function : **dload** loads a program from the disk drive.
dsave saves a program on the disk drive.

Syntax : **dload** "prog. name"
dsave "prog. name"

"prog. name" : Any program name

Utilisation : **dload** and **dsave** are used like load and save. dload can be used inside a program to chain programs together.

Example : **dload** "demo"
dsave "demo"

Note : 1) If you wish to save a program on disk under a name that is already on the directory, you have to use the "@" charater in front of the name.

```
dsave"@demo"
```

2) Complete syntax :

```
dload "prog. name" [, d(n)] [ on u(n')]
dsave "prog. name" [, d(n)] [ on u(n')]
```

header

Function : header formats a blank disk or clear an old disk.

Syntax : header "disk name", d(n) ,i(id)

"disk name" - Any name
d(n) - Drive number (mandatory)
On a 1541, d0
i(id) - Disk ID number - id : any two digits.
i00, i99, iaa, izz

Utilisation : Providing a disk name, a drive number and a disk identification number, **header** formats a blank disk to be ready for use on the drive. Be careful when you perform that instruction on an already used disk as it will erase any previous information on the disk.

After giving the command **header**, the computer will ask you :

are you sure?

If you have any doubt, hit any character but "y" [and (RETURN)] and check your disk. If you answer "y", then the disk is formatted in approximatively 90 seconds.

Example : **header** "master disk", d0, i83

are you sure?y

Note : Complete syntax

header "disk name", d(n) [, i(n')] [on u(n')]

directory
catalog

Function : directory (and **catalog**) gives the contents of the disk.

Syntax : catalog
directory

Utilisation : Both instructions have the same use. They display on the screen the diskette contents. A big improvement over the BASIC 2.0 syntax :

```
load"$",8:list
```

is that using catalog (or directory) will not destroy the program in memory.

Pressing the **SPACE bar** will stop the display of the directory. **SPACE bar** again will restart the listing again.

Example : cA
(Remember that you can abbreviate the commands to 2 or more characters with the last one shifted. See Appendix F)

Note : 1) - Complete syntax

```
directory [ d(n)] [, u(n') ]
catalog [ d(n)] [, u(n') ]
```

2) - If you want a copy of your directory on the printer, you have two solutions.

If the listing is smaller than the screen :

```
catalog : hardcopy
```

If the listing is bigger than the screen :

```
open 4,4:cmd4:catalog:print#4:close4
```

copy - **copy** file

Function : **copy** copies a file.

Syntax : **copy** "file name 1" to "file name2"

"file name1" : Program to be copied.
"file name2" : Name of the copy.

Utilisation : **copy** allows you to make copies of your program on the disk under different names without going thru the memory of the C-64.

Example : **copy** "demo" to "demo-backup"

Notes :1) - The previous example is equivalent to :
dload "demo" : dsave "demo-backup"

but without physically loading the program in memory.

2) - Because MASTER FILE's are composed with different types of files, the COPY instruction does not operate on those files. To copy a MASTER FILE, see utilitary program COPY FILE - Appendix D-9

3) - Complete syntax :

copy "file name1" [,d(n1)] to "file name2" [,d(n2)] [on u(n')]

.....

rename - **rename** file

Function : **rename** renames a file.

Syntax : **rename** "file name 1" to "file name2"

"file name1" : Old name of any file.
"file name2" : New name of any file.

Utilisation : **rename** allows you to change the names of your files. Any file represented by one line on a directory can be modified that way.

Example : **rename** "demo" to "demo-backup"

Notes : 1) - The previous remark on MASTER FILEs applies here too.

2) - Complete syntax :

rename "file name1" [,d(n)] to "file name2" [on u(n')]

scratch - **scratch** file

Function : scratch scratches a disk file.

Syntax : scratch "file name"

"file name" : Program to be scratched.

Utilisation : scratch allows you to destroy from a disk any unwanted file and get the disk space available for other files. It's a non-reversible instruction like HEADER and a scratched file can not be easely recovered by BASIC means.

Example : scratch "demo"
are you sure?y
01,files scratched,01,00

Note - Complete syntax :

scratch "file name" [,d(n1)] [on u(n')]

---

collect

Function : collect cleans up disks.

Syntax : collect "file name 1" to "file name2"

Utilisation : collect frees up the space allocated to improperly closed files on disk and deletes their references from the directory.

Example : collect

Note - 1) - Complete syntax :

collect [d(n)] [on u(n')]

2) - **Do not use this instruction on a disk containing MASTER Files.** (See 6-23).

backup - backup disk

Function : backup copies the whole contents of a disk on another one.

Syntax : backup d(n) to d(n')

d(n) : Source drive number - d0 or d1
d(n') : Object drive number - d1 or d0

Utilisation : backup duplicates the entire contents of a CBM disk on another diskette. This instruction is of no use for 1541 users. It can be used by owners of an IEEE interface compatible with MASTER and dual disk drives such as 2040, 4040, 8050 and 8250.

For the 1541 users, there is a specific back up program available on the MASTER disk allowing to do complete copies of your diskette. (See Appendix D-10)

Example : backup d0 to d1

Note - Complete syntax :

backup d(n) to d(n') [on u(n')]

dopen - disk file **open**
dclose - disk file **close**

Function : **dopen** opens a sequential / relative file for access.
dclose closes opened files.

Syntax : **dopen#** fn,"file name" [,l(ln)]
dclose# fn

fn : logical file number (1 <= l <= 255)
"file name" : Program to be scratched.
l(ln) : Length of the record (relative file)
Ex. : 125, 136

Utilisation : **dopen** activates two types of files. If the length is not given, it opens a **sequential** file. If the length is given, it opens a **relative** file. BASIC 2.0 has the sequential files and just refer to your disk drive manual for more details. We will explain more in page 9-11, the relative files.

A sequential file is automatically opened for READ operations.

A relative file is automatically opened on both READ and WRITE operations.
For WRITE-ONLY operation , add a **",w"** at the end of the instruction.

dclose will evidently close the file corresponding to the logical file number.

Examples:
Opens a sequential file for READ operation.
```
dopen#1,"file 1"
```
Opens a sequential file for WRITE operation.
```
dopen#2,"file 2",w
```
Opens a relative file for READ and WRITE operations.
```
dopen#3,"file 3",135
```
Closes all previous files
```
dclose#3 : dclose#2 : dclose#1
```
or
```
dclose
```

Note - 1) Complete syntax :
```
dopen# fn,"file name" [,l(ln)] [,d(n)] [,on u(n')] [,w]
dclose# fn
```

2) - It is possible to replace an existing file with **dopen**, using the **@** symbol like in **dsave** (cf. 9-3).

3) - dclose without logical file number will close all files.

append

Function : **append** allows to write additional data on a sequential file

Syntax : **append#** fn,"file name"

"file name" : Sequential file

Utilisation : **append** is like a **dopen** except it applies only to sequential files. It activates the file and allows the next record to be added at the end of the sequential file (which you can not do with a DOPEN). **append** allows to expand your sequential file with ease.

Example :
```
append# 2,"file 2"
print# 2,"Hello there"
dclose 2
```

Hello there will be added at the end of the sequential file "file 2".

Note - Complete syntax :

append "file name" [,d(nl)] [on u(n')]

---

concat

Function : **concat** concatenates sequential files.

Syntax : **concat** "file name 1" to "file name 2"

"file name 1" : First sequential file
"file name 2" : Second sequential file

Utilisation : **concat** adds two sequnetial files together. The resulting file is the first one plus the second one, under the name of the first one. The second sequential file is then scratched.

Example : **concat** "file 1" to "file 2"

File 1 is now file1 plus file 2 and file 2 does not exist any more.

Note - Complete syntax :

concat [d(nl),] "file name 1" to [d(n2),] "file name2" [on u(n')]

record

Function : record positions the record pointer in a relative file.

Syntax : record# fn , rn [,b]
fn : logical file number
rn : record number (between 1 and 65535)
b : (optional) - byte pointer inside the record

Utilisation : record is used before **input#, get#, and print#** to position the record pointer inside a relative file. It allows operation like writing the 25 th. record (even if there is no previous record), reading the 11 th. record (direct access). Remember that in order to read the 11 th. record of a sequential file, you have to read the first tenth prior that operation.

The byte pointer b allows to modify, print or read part of a record. (See example 2)

Example :

1) - Create a relative file with 10 records of length 80 characters and write the message 'hello happy taxpayer'

```
10 dopen#1,"@tax file",180
20 for i=1 to 10
30 record#1,(i)
40 print#1,"hello happy taxpayer #"+str$(i)
50 next i
60 dclose#1
```

2) - Read the word 'taxpayer' in the 7 th. record.

```
100 dopen#1,"tax file",180
110 record#1,7,13
120 input#1,na$
130 out na$,5,1
140 dclose#1
150 end
```

Notes :1) - Remember that if a variable is used in the **record** instruction, it must be put between parenthesis. (See line 30 - See also Cf Notes 4 Page 9-2)

2) - With MASTER FILE, the MASTER BASIC random access file instructions, the sequential and relative files, many posibilities are offered to the user and depending for what application the files are used, one system will better than another.

Relative files should be used for simple file with only one index (always numeric with that number between 1 and 65535).

3) - Complete syntax : (Same as previously indicated)

record# fn,"file name", rn [,b]

ds
ds$

Function : **ds** returns an error code corresponding to the last disc operation.

ds$ returns the complete disk status corresponding to the same operation.

Syntax : ds ds$
ds : Reserved numeric variable
ds$: Reserved string variable

Utilisation : ds is a very useful variable indicating at any given moment how is your disk drive. (See Appendix C-2 - for the list of ds,ds$ -error messages). **ds$** gives more details about the disk status.

- The string consists of an error code (same as **ds**), an English message, a track and sector number.

Knowing the type of problems occuring to the disc, the program can give indications on what the operator must do to recover a disk failure. (See example 2)

Examples:

1) - Normal status

```
?ds$                          ?ds
00, ok,00,00                  0
```

2) - The program tries to open a relative file on the disk. Try this program - first with no disk in the drive, and secondly, with a write-protected disk.

```
100 dopen#5,"file",135
110 if ds=0 then goto 150 : rem every thing is OK
120 dclose#1
125 if ds=74 then gosub 200:goto 100
130 dclose#1:out"Other disk problem":stop
150 ...
200 rem error message
210 out "No disk in the disk drive",5,1
220 out"Insert a disk in the drive",6,1
230 out"Press SPACE when ready.",7,1
240 get a$:if a$="" then240
250 return
```

Note - The BASIC 2.0 equivalent of '**?ds$**' is :

```
10 open 15,8,15
20 input#15,a$,b$,c$,d$
30 close15
40 ?a$,b$,c$,d$
```

Function : MASTER includes a simple Machine Language Monitor (MLM) for the machine languages wizards. It allows to look at the memory, at the registers, write simple machine language routines and to save/load them.

Syntax : To access MLM - **sys4 and [RETURN]**

Utilization :

The next display is :

```
b*
      pc  irq  sr ac xr yr sp
);   0005 ea31 30 00 5e 04 f6
)
```

pc - program counter
   - current location of the command being executed
irq - interrupt request address
sr - status register value
ac - accumulator value
xr - x register value
yr - y register value
sp - stack pointer - first empty location on the stack

List of Commands : To be typed after).

r - registers, m - memory contents, g - go

l - load machine code, s - save machine code, x - exit

r - display registers

Syntax :)r - No parameters

That will give the display showed above. Each time you entered MLM, the display register function is called. The function r displays the values of the program counter, interrupt request address, status register, accumulator, x and y registers and stack pointer. Those information give details on what happened last to the microprocessor.

m - display memory contents

Syntax :)m xxxx yyyy
with xxxx - 4-digit HEX number for start address
yyyy - 4-digit HEX number for end address

Syntax Example :)m 0800 0830

Utilization : You can write your machine code by entering now 2-digit HEX number in the locations display in the screen and by hitting [RETURN] after each modified lines.

Example : Enter :

```
new
sys4

b*
      pc  irq  sr ac xr yr sp
);   0005 ea31 30 00 5e 04 f6
)m 0800 0808
                Enter now
):  0800 00 0b 08 0a 00 99 22 31
):  0808 32 22 00 00 00  (The rest has no importance)
)x

list

        10 print"12"     appears.
```

You just entered the HEX code for the program '10 print"12"'

g - goto - Execution of machine code program

Syntax :)g xxxx
with xxxx - 4-digit HEX number for start address

Example :)g 0800

The machine language routine located at the address xxxx will be executed. In our example, that would not work as we have create a BASIC program. The GO command is only for machine code programs.

s - save a machine code program

Syntax :)s "name",01,xxxx,yyyy
)s "0:name",08,xxxx,yyyy
with xxxx - 4-digit HEX number for start address
yyyy - 4-digit HEX number for end address
01 - save on tape
08 - save on disk

Example : **)s "0:mcl-prg",08,2015,2089**

Save on drive 0, disk unit 8, the routine between location 2015 and 2089

l - load in memory a machine code program

Syntax :)l "name",01
)s "0:name",08
with 01 - save on tape
08 - save on disk

Example : **)l "0:mcl-prg",08**

Load from drive 0, disk unit 8, the routine called "mcl-prg".

x - exit from MLM

Syntax :)x

The command x will bring back BASIC and MASTER.

CAUTION

1) - This machine language monitor is a useful add-on to the system. However, it is not a tool for developing complex routines in machine code. Other more powerful softwares are on the market for that purpose.

2) - High-level programming languages like BASIC, MASTER and PASCAL have been invented to free users from the difficulties of having to jungle with registers, interruptions and memory locations. If you start to play with the machine language, using any assembler or MLM-type of programs, you have to expect crashing the system, by giving values to memory locations that the processor will not comprehend. The only thing to do is to switch off your machine and re-boot the system. That is common to all machine language users, so **do not blame MLM and MASTER for any crash that could occur during your experiments in machine code.**

On the other hand, a total crash during the normal use of MASTER (other than MLM) is not probable and you should contact your dealer or Micro Application in such a case.

3) - This chapter is not intented to be a machine language course, but simply a tutorial of the possibilities of MLM. For more information, there are many books available on the subject including :

- Commodore 64 - Programmer's Reference Guide
  - Commodore Computers.
- Programming the PET/CBM - Raeto West
- 6502 Assembly Language Programming
  Osborne / McGraw-Hill

Instructions

clear ln,l,c	- erases ln characters (origin point :l,c)	4-5
clearz n	- erases the nth zone.	4-16
decz n,l,c,ln[,ty][,f$]	- declares the nth zone.	4-11
inz n,a$	- transfers the contents of the nth zone into a$	4-14
out a$,l,c	- displays a$ (origin point :l,c)	4-6
outz n,a$	- displays a$ into the nth zone and contols a$	4-15
reqz n	- requires an answer in the nth zone	4-13
rev l,c,lg,ln[,co]	- reverses video on part of the screen	4-8
revz n	- reverses video mode in the nth zone	4-17
sclear	- clears a screen page	4-22
scopy	- transfers a screen page to the screen	4-19
screen sc,br [,cr]	- defines screen color	4-9
scroll l,c,ln,lg,ty	- scrolls up (down) a part of the screen	4-7
sexch	- exchanges a screen for one in memory	4-21
sload 8,"0:name"	- loads a screen page in memory (screen)	4-25
sreset pm	- resets a screen page	4-23
ssave 8,"0:name"[,n][,n2ton3]	- saves a screen page on disk with zones	4-24
sset	- transfers all controls to the screen	4-19
sset pm	- transfers all controls to the pmth screen page	4-19
tcol lg,l,c	- draws a col. of height lg (origin l,c)	4-4
tline ln,l,c	- draws a line of length ln (origin l,c)	4-3

Reserved variables

zo	- ASCII code for the return key in request mode (decz)

Non-Reserved Variables (Programmer's Choice)

a$	- string - transfer
c	- column number (x-coordinates)
f$	- pusing format
l	- line number (y-coordinates)
lg	- width (or height) of a window (zone) - number of lines
ln	- length of a window (zone) - number of columns
n1,n2,n3,n	- zone identification number
pm	- screen page (in memory) id number
ty	- formatting type (decz) - scrolling type (scroll)

Intructions

pclear ln,l,c	- erases ln characters (point of origin)	5-8
pclearz n	- erases the nth zone	5-8
pclose	- closes the print page	5-5
pcreate du,"dn:"+np$,nl,nc, hp,fp,mr	- declares the print page np$	5-3
pdecz n,l,c,ln[,ty][,f$]	- declares the nth zone	5-6
perase	- erases a complete print page	5-9
popen du,"dn:"+np$	- opens the print page np$	5-4
pout a$,l,c	- displays a$ (origin point:l,c)	5-7
poutz n,a$	- displays a$ in the nth zone	5-7
pprint np	- prints the page thru the np channel	5-10

Reserved Variables

None

Non-Reserved Variables (Programmer's Choice)

a$	- string (transfer)
c	- column number (x-coordinates)
dn	- drive number
du	- floppy unit logical number (usually 8)
f$	- pusing format
fp	- number of blank lines at top of page (0 to 255)
hp	- number of blank lines at bottom of page (0 to 255)
l	- line number (y-coordinates)
ln	- length of a zone
mr	- left margin (0,255)
n	- zone identification number (0,127)
nc	- max. number of columns for a print page
nl	- max.number of lines for a print page
np	- printer channel number
np$	- page name
ty	- format type

Instructions

creatst a$,ln[,chr$(ca)]	- creates a string called a$	6-29
getst a$;ty,b$,pt,ln[;..	- unpacks a string (record)en$	6-31
j nf, en$	- adds a record without updating index	6-10
i **se** nf	- closes a file	6-20
idata nf,mk$,en$	- reads a file sequentially	6-19
idelete nf,cl$	- deletes a record using its primary key	6-13
iexist nf,cl$	- tests if a record exists	6-12
inext nf,mk$, en$	- reads in ascending order	6-15
inext-nf,mk$,en$	- reads in descending order	6-15
iopen nf,du,"dn:"+fn$	- opens a file	6-8
iread nf,cl$,en$	- reads a record by its primary key	6-11
ireset	- general reset of MASTER	6-22
irestore nf	- initializes a sequential search	6-19
istart nf,cl$,mk$, en$	- starts a search in ascending order	6-15
istart-nf,cl$,mk$,en$	- starts a search in descending order	6-15
iupdate nf,en$	- updates a record	6-14
iupgrade nf	- saves all index tables	6-20
ivalidate nf [;n]	- validates all IADD records(batch)	6-18
iwrite nf,en$	- writes a record	6-9
putst a$;ty,b$,pt,ln[;.]	- packs a string or a collection of strings)	6-30

Reserved Variables

l.	- length of the record key
lr	- length of the record
ni	- number of records in the index tables
nr	- number of records in the file
ok	- status variable

Non-Reserved Variables (Programmer's Choice)

cl$	- primary key
dn	- drive number (0/1)
du	- floppy unit logic number (8)
en$	- record
fn$	- file name
mk$	- retrieval mask
n	- zone number for automatic counter display
nf	- file number
pt	- pointer in a string (packing/unpacking)
ty	- packing type

Instructions

datepack	- controls and packs a date	7-9
dateunpack	- unpacks a date	7-9
goto lb	- computed goto	7-4
gosub lb	- computed gosub	7-4
hunt chr$(ca),a$,pn	- searches a character within a string : presence	7-11
hunt-chr$(ca),a$,pn	- searches a character within a string : absence	7-11
inblock tr,se[,nd][,nu]	- transfers a disk block to a buffer	7-7
madd nl$,n2$,n3$	- multiprecision addition (n3$=nl$+n2$)	7-2
mdiv nl$,n2$,n3$	- multiprecision division (n3$=nl$/n2$)	7-3
mmul nl$,n2$,n3$	- multiprecision multiplication (n3$=nl$*n2$)	7-3
msub nl$,n2$,n3$	- multiprecision subtraction (n3$=nl$-n2$)	7-2
nolist "prog",d	- saves in nolist mode	7-8
putblock tr,se[,nd][,nu]	- transfers a buffer on disk track	7-6
putbuf a$,pn	- transfers a string into a disk buffer	7-6
takbuf a$,pn,ln	- transfers a buffer into a variable	7-7
uplow a$	- upper/lower case letter inversion	7-5

Reserved Variables

da	- resulting binary date (date packing)
da$	- working date (dd/mm/yy)
dr$	- reference date (dd/mm/yy)
ok	- status variable

Non-Reserved Variables (Programmer's Choice)

a$,b$	- strings (transfer)
ca	- ASCII value of a character
d	- drive (d0 or d1)
lb	- line number
ln	- length of transferred characters
nl$,n2$,n3$	- strings containing multiprecision numbers
nd	- drive number (0 or 1)- (0 if not defined)
nu	- floppy unit logical number (usually 8)
pn	- pointer for the disk buffers
se	- sector number (disk)
tr	- track number (disk)

Instructions

auto n	: numbers the lines automatically	8-2
beep	: sounds bell	8-12
delete l1-l2	: deletes program lines (like list)	8-3
dump	: lists variables with values	8-5
error	: locates an error within a line	8-6
find <deli><string> <deli>	: finds a string within a program	8-7
hardcopy	: screen dump to printer	8-8
if <c> **then** <i1>**:else** <i2>	: structured programming instruction	8-13
plot x,y	: plots a point on screen	8-9
pusing#[ ],F$,P1,P2..	: formatted printing	8-11
renu l1,l2,l3	: renumbers a program automatically	8-4
reset x,y	: erases a plot	8-9

Reserved variables

None.

Non Reserved Variables (Programmer's Choice)

c	: origin point (colum number)
<c>	: condition (ex: A = B)
d	: drive number (d0 or d1)
<deli>	: delimiter for find function
f$	: format for pusing
<i1> <i2>	: instructions
l	: origin point (line number)
lc	: height of the window (number of lines)
ln	: length (number of columns)
l1,l2,l3	: program lines numbers
n	: increment for the auto numbering
<string>	: characters to be searched in a program
x,y	: coordinates of a point to draw
#	: optional **with** - editing on printer **without** - editing on screen

Instructions

append# fn, "file" [,d(n)]	: Adds records to sequent. file	9-10
backup d(n) to d(n')	: Copies a disk (dual-disk drive)	9-8
catalog [d(n)]	: Gives the contents of a disk	9-5
collect [d(n)]	: Cleans a disk	9-7
concat "file1" to [d(n),] "file2"	: Concatenates two sequent. files	9-10
copy "file1" [,d(n)] to "file2"	: Copies a file	9-6
dclose fn	: Closes a file	9-9
directory [d(n)]	: Gives the contents of a disk	9-5
dload "prog" [,d(n)]	: Loads a program from disk	9-3
dopen# fn,"file" [,l(ln)]	: Opens a file	9-9
dsave "prog" [,d(n)]	: Saves a program on disk	9-3
header "disk",d(n), i(id)	: Formats a disk	9-4
record# fn, rn [,b]	: Accesses a relative file	9-11
rename "file1" [,d(n)] to "file2"	: Renames a file	9-6
scratch "file" [,d(n)]	: Deletes a file	9-7

Reserved variables

ds	: Disk error code	9-12
ds$	: Disk message status	9-12

Non Reserved Variables (Programmer's Choice)

b	: Pointer within a relative record
d(n), d(n')	: Drive number (d0 or d1)
"file" "file1" "file2"	: File name
fn	: File identification number
i(id)	: Disk identification number
"prog"	: Program name
rn	: Record number

MASTER SCREEN

zo - ASCII code for an exit in Request Mode
(see decz : 4-10,11)

1 to 26	- CTRL / A to Z
9	- TAB
13	- RETURN
17	- CRSR DOWN
18	- RVS
64	- @
65 to 90	- A to Z
133 to 140	- F1 to F8
141	- SHIFT + RETURN
145	- CRSR UP
146	- OFF (SHIFT+RVS)

MASTER FILE

lk - length of the record key (2<=lk<=30)
lr - length of a record (2<=lr<=254)
ni - number of active records (that can be accessed)
nr - total number of records (including deleted ones)
ok - status variables

0	- OK execution
1	- No corresponding record to the given key
10	- (see MASTER BASIC)
64	- file full
128	- file not found
255	- end of file

MASTER BASIC

da - Packed value of da$ in binary (2 bytes)
dr$ - reference date
da$ - date to be packed
01 < = DD < = 31
01 < = MM < = 12
00 < = YY < = 99

ok - status variable 10 : packing or date control
non authorized-bad date
other than 10 : see MASTER FILE

BASIC 4.0

ds - Disk error code (See 9-12 and C-2)
ds$ - Disk status

Error messages generated by MASTER :

ARE YOU SURE? : This message is displayed each time a HEADER or a SCRATCH instruction is used. If the answer is different from **y** or **yes**, then the instructions will not be executed. (see 9-4, 9-7)

BAD DISK : 1) In the **HEADER** instruction, the disk ID number has been omitted.

2) The same instruction has not been executed because of a disk problem. (See 9-4)

BAD FORMAT : 1) **decz** using the p-type : the pusing defined does not correspond to the variables used - length problems (see 4-10)

2) The format of the PRINTUSING is not correct.

3) **packing/unpacking** b-type
There is a contradiction in the putst (floating point number) within the parameters - (ln : always 5 bytes) (see 6-32)

DRIVE FORGOTTEN : **iopen** : drive number has not been given (see 6-9)

ELSE WITHOUT IF : There are more **else** than **if.** (see 9-16)

MEMORY OVERFLOW : MASTER can control 15 pages. If more entries are declared, this message appears (see 6-8)

NO SCREEN : A **sreset** has been performed on a non-existing page. (see 4-22)

NON-EXISTENT ZONE : A **clearz, outz** or **revz** has been performed on a non-declared zone (see 4-9,4-10).

OUT OF FORMAT : Overflow in packing (see 6-32, 33)

OUT OF PAGE OR SCREEN : coordinates of the starting point or length or height are incorrect in the print or screen generator instructions.

OUT OF STRING : Packing/unpacking receiving string too short (see 6-29,6-30,6-31)

RECORD FORMAT : **iwrite** or **iadd** (see 6-9, 6-10) - Length of record different than defined in RESERVE FILE.

Error messages generated by BASIC 4.0 DOS through ds and ds$:

	Error Number	Error Message	Track	Sector
Status Message	00	OK	00	00
	01	FILES SCRATCHED	# files	00
Read Errors	20	READ ERROR (Block header not found)	T	S
	21	READ ERROR (No synch character)	T	S
	22	READ ERROR (Data block not present)	T	S
	23	READ ERROR (Checksum err. / data block)	T	S
	24	READ ERROR (Byte decoding error)	T	S
	27	READ ERROR (Checksum err. / header)	T	S
Write Errors	25	WRITE ERROR (Write/verify error)	T	S
	26	WRITE PROTECT ON	T	S
	28	WRITE ERROR (Long data block)	T	S
	29	DISK ID MISMATCH	T	S
Syntax Errors	30	SYNTAX ERROR (General Syntax)	00	00
	31	SYNTAX ERROR (Invalid command)	00	00
	32	SYNTAX ERROR (Long line)	00	00
	33	SYNTAX ERROR (Invalid file name)	00	00
	34	SYNTAX ERROR (No file given)	00	00
	39	SYNTAX ERROR (Invalid DOS command)	00	00
	50	SYNTAX ERROR (Record not present)	00	00
	51	SYNTAX ERROR (Overflow in record)	T	S
	52	SYNTAX ERROR (File too large)	T	S
File Errors	60	WRITE FILE OPEN	00	00
	61	FILE NOT OPEN	00	00
	62	FILE NOT FOUND	00	00
	63	FILE EXISTS	00	00
	64	FILE TYPE MISMATCH	00	00
	65	NO BLOCK	T	S
	66	ILLEGAL TRACK AND SECTOR	T	S
	67	ILLEGAL SYSTEM TRACK AND SECTOR	T	S
System Errors	70	NO CHANNEL	00	00
	71	DIR ERROR	00	00
	72	DISK FULL	00	00
	73	DOS MISMATCH	00	00
	74	DRIVE NOT READY	00	00

Notes :

For more details on these messages and BASIC 4.0 in general, please consult the following references :

- User's Reference Manual - Com. Basic 4.0 - Part# 321604 - CBM
- Disk System User Reference Manual- - Part# 320972 - CBM
- Pet/CBM Personal Computer Guide - Osborne/Donahue - Osborne
- Any Commodore disk drive manual from the 2031 to the 9090

OVERVIEW

MASTER includes several utilities which are :

- either directly included in the program,
- or in the form of a BASIC program, saved in "nolist" and are called with an overlay, the parameters being transfered through a sequential file called "Transfert".

Utilities included in the program :

To call these utilities, use the "uti" command followed by a letter. This command releases an overlay mechanism where the utility replaces the multiprecision and the printing generator sections of MASTER.

Calling the latter two functions, a reverse overlay is released, making the system totally transparent to the user. To do this, MASTER "uti" modules must be present on the diskette. (See 3-2, 3-3)

utib : Display contents of a disk buffer in HEX
utic : Gives the MASTER serial number
utid : Dump of a file
utig : Merge of a sequential file into a MASTER file
utih : Dumps a file in HEX
utik : Reads what disk drive is used
util : Tells MASTER what disk drive is being connected

BASIC utilities :

All these utilities have to be loaded either by a dload or an overlay.

RESERVE FILE	- MASTER file definition
RESERVE 1 & 2	- Variations of RESERVE FILE
REGEKEY	- Regeneration of the Index tables from a MASTER FILE
COPY FILE	- Copies a MF from one drive to another recovering all the room wasted by idelete
1541 BACKUP	- Copies entire disk on a 1541 single drive.

utib

Function : **utib** reads a buffer in HEX.

Syntax : **utib** [ch]
ch : peripheral number for the edition
(optional - if no ch, screen).

Utilisation : **utib** dislays the contents of a buffer, for example, after an **inblock**, to check data within the buffer.

Example :

```
10 open 1,3
20 inblock 18,1,0
30 utib 1
```

---

utic

Function : **utic** gives the serial number of MASTER with its release.

Syntax : **utic** be$
be$: receiving string

Utilisation : To check release and serial number.

Example :

```
utic a$
out a$,10,1

64 v1.1 dev #10983-0
```

Notes :

This instruction is very useful in the case of protected run-time. (See Appendix G)

utid

Function : **utid** dumps a file to a channel (printer).

Syntax : **utid** nf,mk$,ch, d(or n), p(or r) [,n]

nf : MASTER file number,
mk$: retrieval mask,
ch : existing channel (in the example : the printer),
d or n : d : reading in "idata" order
n : reading in "inext" order
r or p : sets an "irestore" or "istart" at the beginning of the file,
n : displays, if necessary, a counter in a predefined zone.

Utilisation : **utid** controls the contents of a MASTER file, during the transfer to other files or peripherals (back-up, for example).

Examples : Printing on a printer of a MASTER file.

```
10 open 4,4 : iopen 1,8,"0:file"
20 decz 6,10,10,10
30 utid 1,"",4,n,r,6
```

Transfer of a MASTER file into a sequential file

```
10 dopen #10,"seq.file",w
20 iopen 1,8,"0:file"
30 decz 3,10,10,4,n
40 utid 1,"",10,d,r,3
50 iclose 1 : dclose#10
```

Note : All these operations can be performed with a mask as previously defined (see "istart", "inext").
chr$ (13) separates the records from each other.

utig

Function : utig appends a sequential file to a MASTER file.

Syntax : utig nf,mk$,ch, w(or a) [,n]

nf : MASTER file number
mk$: retrieval mask
ch : exit channel
w or a w - use of iwrite
a - use of iadd
n : (optional)-allows the display of a counter in a predefined zone.

Utilisation : Appends a sequential file previously dumped to a MASTER FILE

The MASTER FILE must be created previously with the correct record length. It can be blank or active. In the latter case, watch for duplicates when working in "iadd" (see the remarks about this command).

Example :

```
10 dopen#5,"seq.file",r
20 iopen 1,8,1,"0:file2"
30 decz 99,10,10,4,n
40 utig 1,"",5,w,99
50 iclose 1 : dclose#5
```

utih

Function : dumps a file in HEX.

Syntax : **utih** nf,mk$,ch, d(or n) , p(or r) [,n]

nf : MASTER file number,
mk$: retrieval mask,
ch : existing channel (in the example : the printer),
d or n : d : reading in "idata" order
n : reading in "inext" order
r or p : sets an "irestore" or "istart" at the beginning of the file,
n : allows,if necessary, the display of a counter in a predefined zone.

Utilisation : **utih** similar to utid (normal dump).

The only difference is the kind of characters obtained. The HEX form is useful if the file has been packed.

Example : (see utid : D-3)

util

Function : util tells MASTER what kind of disk drive is being used.

Syntax : util ty, du

ty : disk type - 1541,4040,8050,8250,9060,9090
du : disk drive logical number 8<=du <=14

Utilisation : A look at the example should explain it clearly. Only one util can be declared at a time. If you use a second **util** during a program, the previous one will be erased.

Example : Working with a 1541, logical number 8.

```
10 util 1541, 8
```

Note:

1) The system automatically defaults to a 1541 disk drive and device 8.

2) For a 2031 drive, enter 4040.

3) If the **util** is incorrect, ok is set to 50. The following program sets up MASTER automatically for the connected disk drive.
(Can be part of the BOOT program.)

```
10 du=8
20 for i=0 to 5
30 read ty:util ty, du
40 if ok=0 then 200
50 next i
60 out"Problem with the disk drive",1,1:beep:stop
70 end
80 data 1541,4040,8050,8250,9060,9090
```

---

utik

Function : utik reads MASTER to determine what disk drive is in use.

Syntax : utik
without parameters. The characteristics of the disk drive will be given in the following variables.
ty - disk unit type
t - track number for beginning of the directory
s - sector number for beginning of the directory

Utilisation : See example.

Example : utik : ? ty,t,s

Note : Can be used only if **util** has been used.

Reserve file - MASTER file definition (see 6-2 etc)

Reserve 1 and 2 - Programmed Utilisation of RESERVE FILE

- **reserve 2** is a module that can be automatically called in overlay by a BASIC program; the parameters of the file to be created being passed through a file called "transfert",

- **reserve 1** is a example program of the use of "reserve 2", which creates a MASTER FILE called 'employee' within an application program (transparent for the user).

Example : reserve 1 - To create a file within a program :

```
10 iopen 1,8"0:employee":if ok=1 then 150
20 dopen#10,"@transfert",w: rem 'transfert' file creation
30 print#10,"file"          :rem  file name
40 print#10,"0"             :rem  drive for data
50 print#10,"0"             :rem  drive for the indexes
60 print#10,1               :rem  level 1 or 2
70 print#10,nbrec           :rem  maximum number of records
80 print#10,lrec            :rem  record length
90 print#10,lkey            :rem  key length
100 print#10,kdeb           :rem  key position
110 print#10,"prg"          :rem  prog. to load after execution
120 print#10,0              :rem  drive number for next program
130 dclose#10
140 dload "reserve 2"
150 ...
```

Regekey - Index Table Regeneration

Function : This program regenerates index files, from the data file :

- if a file named "transfert" does not exist, then **regekey** works in direct mode,

- if the file exists, then the parameters of the file to regenerate (name and drive) are taken from "transfert".

Example :

```
10 dopen#10,"transfert",w:    rem  'transfert' file creation
20 print#10,"file":           rem  name of file to regenerate
30 print#10,dd$:              rem  "d file" drive
40 print#10,"prg":            rem  prog to be executed after
45 :                          rem  after the regeneration.
50 print#10,0:                rem  "prg" drive
60 dclose#10
70 dload "regekey"
```

Notes : A MF file can be copied by the "copy" command (copy the *** d file) of BASIC 4.0; and then recreate the new index table by **"regekey"**.

A file created with **regekey** is not validated. The records must be validated by "ivalidate" to get key access, i.e., create index table.

Copy File - Back up MF files

Function : This program has two functions. It can copy MF files (file + index) from a disk to another one, from a disk drive to another one, from a disk unit to another disk unit. It can also reorganize a file by expanding the size of the file, modifying the record size, the key size and position.

Utilization : The following is a hard-copy of the screen as used in **copy file**.

```
-------------------------------------
    SOURCE FILE        OBJECT FILE
--------------------+----------------
                    !
name employee       !name
drive            0  !drive
unit             8  !unit
                    !
key drive           !key drive
level               !level
                    !
# records           !# records
record length       !record length
key length          !key length
key position        !key position
                    !
                    !memory          b  (# bytes)
                    !blk record      s  (# sectors)
                    !blk index       s  (# sectors)
                    !
                    !total           s  (# sectors)
--------------------+----------------
modification record length :

execution y/n :   write or add w/a :
```

After loading the program and place the SOURCE file disk in the drive, the program will ask you the name of the file, the drive number and unit. It will then display the parameters of the file on the left part of the screen. You can now enter the parameters of the new file on the right. If you are just copying the file, just hit [**RETURN**] after each line. You can modify the file name, the drive number and unit, the number of records, the record length, the key length and the key position.

If you modify the length of the record, **copy file** will ask you where you you want the length modification to be realized. In case of a record expansion, it will ask you for a string to insert.

It will finally ask you if you want the execution of the copy function and the method to do so, that is using **iwrite** (See 6-9) or **iadd** (See 6-10).

Notes :

When copy file is executed, all the space used by deleted records (See **idelete** 6-13) is restored and can be used again.

1541 backup

Function : 1541 backup provides a program allowing a 1541 single disk drive user to create backup of its disks.

Utilization : Just load the program and run it. You will have first to insert a new disk that will be refered as the **OBJECT** disk and will be automatically formatted. The operation takes 90 seconds to perform. Then put the **SOURCE** disk in the drive and hit **[SPACE]**.

The Block Allocation Map (BAM) of the **SOURCE** disk is now read. That allows the program to know what blocks of the disk are effectively used, making the backup more efficient. The copying is done by 50 blocks at a time. A counter indicates the number of blocks left to be copied, and messages tell you when to change the disks.

Notes :

1 - 1541 BACKUP makes copies of your original MASTER disk by copying all relevant programs and files. So, you can make all the copies you want of your programs and files and use them with the MASTER RUN-TIME version.

2 - If the diskette that you are copying has bad sectors, 1541 BACKUP will stop the backup and will indicate the bad sector. In that case, try using other method to copy your files.

- MASTER file : COPY FILE utility (See D-9)
- Program : DLOAD"NAME",8 - change disk and DSAVE"NAME",8
- Screen Page : SLOAD"NAME",8 - change disk and SSAVE"NAME",8
- etc ...

OVERVIEW :

These are some examples of programs using MASTER and some demonstrations of MASTER's capabilities. They will give some ideas as how to get the best out of MASTER. They are not protected in NOLIST mode, and can be listed.

DEMOPRG

This program is an example of writing and reading a MASTER file. The program will create randomly 25 records, will write them, read them in ascending and descending order, in creation order and will finally scratch the file, while showing the instructions used to do so. MAKE A COPY OF YOUR MASTER ORIGINAL DISK USING '1541 BACKUP' (See D-10) AND USE THAT COPY TO DEMONSTRATE 'DEMOPRG'.

DEMOSCREEN

This program is an example of the use of MASTER SCREEN instructions including tracing lines and columns, definition of zones, display with print using, color manipulation and scrolling of data.

PROSCREEN 64

This program is a screen-page editor, allowing the user to easely design screen pages and save them for future use. It is not only a demonstration but a very useful utility. It comes with a HELP page reproduced below.

HELP

This is a screen page to explain the use of PROSCREEN 64. It can be display during the execution of PROSCREEN with CTRL ^ or in direct mode by typing **:sload 8,"help" and [RETURN]**

```
Proscreen is an editor to create screen
pages with your MASTER 64. All Commodore
keys are available with the following
add-on functions :

f1: draws a line
f2: draws a line until the next column
f3: draws a column
f4: draws a column until the next line
f5: draws a frame
f6: clears a vertical line
f7: screen colors
f8: "painting" functions

CTRL <- : disk commands
CTRL ^  : help screen

     +--------------------------+
     ! To go back to the Editor !
     !                          !
     !      Press any key       !
     +--------------------------+
```

List of all the instructions with short forms.

Instru.	Short Form		Page	Instru.	Short Form		Page
append	aP		9-10	msub	mS		7-2
auto	aU		8-2	nolist	nO		7-8
backup	bA		9-8				
beep	bE		8-12	off	oF		8-14
catalog	cA		9-5	out	oU		4-6
clear	cL		4-5	outz	oUz		4-15
clearz	cLz		4-16	pclear	pclE		5-8
collect	coL		9-7	pclearz	pclEz		5-8
concat	conC		9-10	pclose	pcL		5-5
copy	coP		9-6	pcreate	pC		5-3
creatst	cR		6-29	pdecz	pD		5-6
datepack	dA		7-9	perase	pE		5-9
dateunpack	dateU		7-9	plot	pL		8-9
dclose	dC		9-9	popen	pO		5-4
decz	dE		4-11	pout	poU		5-7
delete	deL		8-3	poutz	poUz		5-7
directory	diR		9-5	pprint	pP		5-10
dload	dL		9-3	pusing	puS		8-11
dopen	dO		9-9	putblock	putbL		7-6
dsave	dS		9-3	putbuf	putB		7-6
dump	dU		8-5	putst	pU		6-30
else	eL		8-13	record	reC		9-11
error	eR		8-6	rename	reN		9-6
find	fI		8-7	renu	renu	(*)	8-4
getst	gE		6-31	reqz	rE		4-13
gosub	goS		7-4	reset	resE		8-9
goto	gO		7-4	rev	rev	(*)	4-8
hardcopy	hA		8-8	revz	revz	(*)	4-17
header	hE		9-4	sclear	scL		4-22
hunt	hU		7-11	scopy	sC		4-20
iadd	iA		6-10	scratch	scrA		9-7
iclose	iC		6-20	screen	scrE		4-9
idata	idA		6-19	scroll	scR		4-7
idelete	iD		6-13	sexch	sE		4-21
iexist	iE		6-12	sload	sL		4-25
if	if	(*)	8-13	sreset	sR		4-23
inblock	inB		7-7	ssave	sS		4-24
inext	iN		6-15	sset	ssE		4-19
inz	inz	(*)	4-14	takbuf	tA		7-7
iopen	iO		6-8	tcol	tC		4-4
iread	iR		6-11	then	tH		8-13
ireset	ireS		6-21	tline	tL		4-3
irestore	iresT		6-19	trace	tR		8-14
istart	iS		6-15	uplow	uP		7-5
iupdate	iU		6-14	utib	uTb		D-2
iupgrade	iupG		6-20	utic	uTc		D-2
ivalidate	iV		6-18	utid	uTd		D-3
iwrite	iW		6-9	utig	uTg		D-4
madd	mA		7-2	utih	uTh		D-5
mdiv	mD		7-3	utik	uTk		D-6
mmul	mM		7-3	util	uTl		D-6

(*) - Note : These instructions do not have short forms.

Comparison between the different MASTER 64 versions.

There are three versions of MASTER 64, two of them being on your diskette.

- These 3 programs constitute **MASTER 64 Development**

  m64.boot

  uti64I13d-a
  uti64I13d-b

That version can not be copied and has to be loaded from the original disk. It allows the loading, saving, writing, modifying, and executing of programs. It also has a NOLIST mode.

- These 5 programs constitute **MASTER 64 Unprotected Run Time**

  loaderu
  master64I10u-a
  master64I10u-b
  uti64I10u-a
  uti64I10u-b

That Run Time version can be copied on disk. It allows the loading, saving, running of BASIC / MASTER programs, but does not allow program listings , new program design and does not run NOLIST programs.

A second run-time version is available that runs with an electronic protection key. The main purpose is to enable the application designer to protect its work. The package is called **UTI-25** and is composed of 25 protection run-time keys having the same number and a disk with the MASTER 64 Protected Run-time Version working with those keys.

For an effective protection, have your program check the key number using the **utic** instruction (See Appendix D-2) and save them in NOLIST mode. Listing and executing the program becomes impossible without that particular Run-Time key.

A certificate is provided with the UTI-25 package to insure that the Run Time key number is unique to this package. If you wish more keys with the same number, join your certificate with your order to guarantee that your number is unique.

For more details, contact your local dealer or Micro Application.

Now that you are using **MASTER 64**, we would appreciate your comments concerning the product quality and any problems which you have encountered. Your comments will assist **MICRO APPLICATION** in improving its products and services.

If you wish a reply and/or if you want to be on the **MICRO APPLICATION** mailing list, please fill in your name and address in the space below:

NAME :
ADDRESS :
:
CITY : STATE: ZIP:

MASTER 64

Which functions do you find most useful?

Which functions do you find least useful?

What improvements would you like to see?

For what kind of applications do you use **MASTER 64**?

Have you found any software problems with **MASTER 64**?

DOCUMENTATION

What is your opinion of this manual?

Strong points?

Weak points?

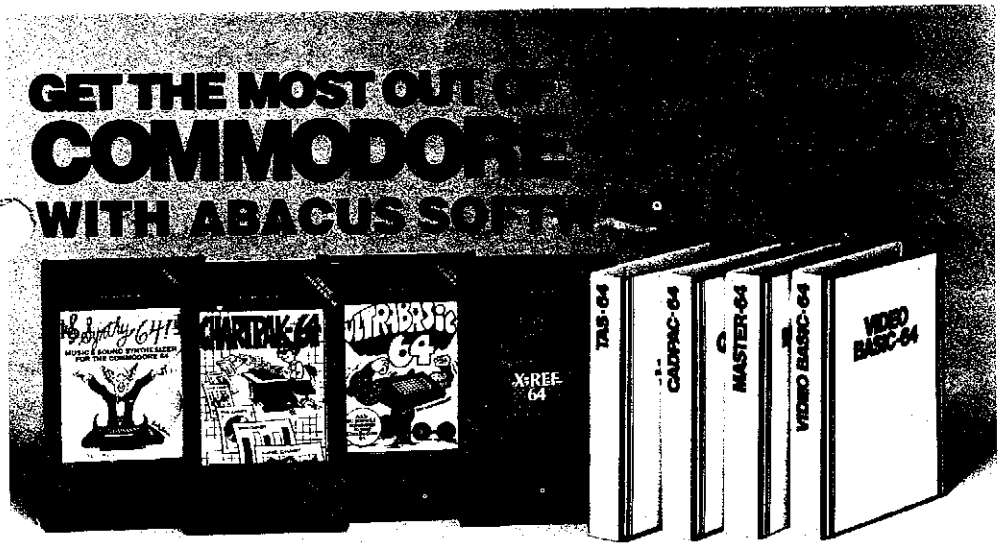
NOTES

Any additional comments--

Mail to:
ABACUS Software
P.O. Box 7211
Grand Rapids, MI 49510

We will forward your comments to **MICRO APPLICATION.** Thank you.

GET THE MOST OUT OF YOUR COMMODORE WITH ABACUS SOFTWARE



XREF-64 BASIC CROSS REFERENCE

This tool allows you to locate those hard-to-find variables in your programs. Cross-references all tokens (key words), variables and constants in sorted order. You can even add you own tokens from other software such as ULTRABASIC or VICTREE. Listings to screen or all ASCII printers.

DISK $17.95

SYNTHY-64

This is renowned as the finest music synthesizers available at any price. Others may have a lot of onscreen frills, but SYNTHY-64 makes music better than them all. Nothing comes close to the performance of this package. Includes manual with tutorial, sample music.

DISK $27.95 TAPE $24.95

ULTRABASIC-64

This package adds 50 powerful commands (many found in VIDEO BASIC, above) - HIRES, MULTI, DOT, DRAW, CIRCLE, BOX, FILL, JOY, TURTLE, MOVE, TURN, HARD, SOUND, SPRITE, ROTATE, more. All commands are easy to use. Includes manual with two-part tutorial and demo.

DISK $27.95 TAPE $24.95

CHARTPAK-64

This finest charting package draws pie, bar and line charts and graphs from your data or DIF, Multiplan and Busicalc files. Charts are drawn in any of 2 formats. Change format and build another chart immediately. Hardcopy to MPS801, Epson, Okidata, Prowriter. Includes manual and tutorial.

DISK $42.95

CHARTPLOT-64

Same as CHARTPACK-64 for highest quality output to most popular pen plotters.

DISK $84.95

DEALER INQUIRIES ARE INVITED

CADPAK-64

This advanced design package has outstanding features - two Hires screens; draw LINEs, RAYs, CIRCLEs, BOXEs; freehand DRAW; FILL with patterns; COPY areas; SAVE/RECALL pictures; define and use intricate OBJECTS; insert text on screen; UNDO last function. Requires high quality lightpen. We recommend McPen. Includes manual with tutorial.

DISK $49.95 **McPen lightpen $49.95**

MASTER 64

This professional application development package adds 100 powerful commands to BASIC including fast ISAM indexed files; simplified yet sophisticated screen and printer management; programmer's aid; BASIC 4.0 commands; 22-digit arithmetic; machine language monitor. Runtime package for royalty-free distribution of your programs. Includes 150pp. manual.

DISK $84.95

VIDEO BASIC-64

This superb graphics and sound development package lets you write software for distribution without royalties. Has hires, multicolor, sprite and turtle graphics; audio commands for simple or complex music and sound effects; two sizes of hardcopy to most dot matrix printers; game features such as sprite collision detection, lightpen, game paddle; memory management for multiple graphics screens, screen copy, etc.

DISK $59.95

TAS-64 FOR SERIOUS INVESTORS

This sophisticated charting system plots more than 15 technical indicators on split screen; moving averages; oscillators; trading brands; least squares; trend lines; superimpose graphs; five volume indicators; relative strength; volumes; more. Online data collection DJNR/S or Warner. 175pp. manual. Tutorial.

DISK $84.95

FREE CATALOG Ask for a listing of other Abacus Software for Commodore-64 or Vic-20

DISTRIBUTORS

Great Britain:
ADAMSOFT
18 Norwich Ave.
Rochdale, Lancs.
706-524304

Belgium:
Inter. Services
AVGuillaume 30
Brussel 1160, Belgium
2-660-1447

France:
MICRO APPLICATION
147 Avenue Paul-Doumer
Rueil Malmaison, France
1732-9254

New Zealand:
VISCOUNT ELECTRONICS
306-308 Church Street
Palmerston North
63-66-696

West Germany:
DATA BECKER
Merowingerstr 30
4000 Dusseldorf
0211/312085

Sweden:
TAL TRADING
PO 516
34300 Almhult
476-12304

Australia:
CW ELECTRONICS
416 Logan Road
Brisbane, Queens
07-397-0806

Commodore 64 is a reg. T.M. of Commodore Business Machines

AVAILABLE AT COMPUTER STORES, OR WRITE:

Abacus Software

P.O. BOX 7211 GRAND RAPIDS, MICH. 49510

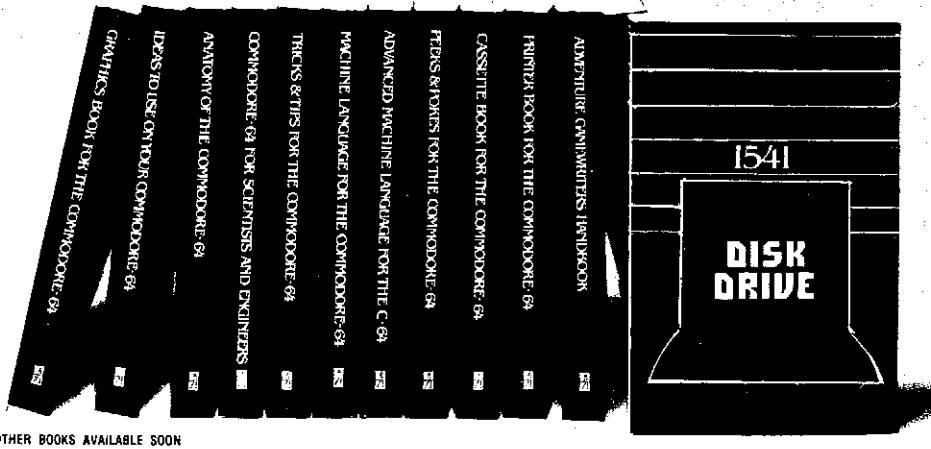
For postage & handling, add $4.00 (U.S. and Canada), add $6.00 for foreign. Make payment in U.S. dollars by check, money order or charge card. (Michigan Residents add 4% sales tax).

VISA MasterCard

FOR QUICK SERVICE PHONE 616-241-5510

FOR COMMODORE-64 HACKERS ONLY!

The ultimate source for Commodore-64 Computer information



OTHER BOOKS AVAILABLE SOON

THE ANATOMY OF THE C-64

is the insider's guide to the lesser known features of the Commodore 64. Includes chapters on graphics, sound synthesis, input/output control, sample programs using the kernal routines, more. For those who need to know, it includes the complete disassembled and documented ROM listings.

ISBN-0-916439-00-3 300pp $19.95

THE ANATOMY OF THE 1541 DISK DRIVE

unravels the mysteries of using the misunderstood disk drive. Details the use of program, sequential, relative and direct access files. Include many sample programs - FILE PROTECT, DIRECTORY, DISK MONITOR, BACKUP, MERGE, COPY, others. Describes internals of DOS with completely disaddembled and commented listings of the 1541 ROMS.

ISBN-0-916439-01-1 320pp $19.95

MACHINE LANGUAGE FOR C-64

is aimed at those who want to progress beyond BASIC. Write faster, more memory efficient programs in machine language. Text is specifically geared to Commodore 64. Learns all 6510 instructions. Includes listings for 3 full length programs: ASSEMBLER, DISASSEMBLER and amazing 6510 SIMULATOR so you can "see" the operation of the '64.

ISBN-0-916439-02-X 200pp $14.95

TRICKS & TIPS FOR THE C-64

is a collection of easy-to-use programming techniques for the '64. A perfect companion for those who have run up against those hard to solve programming problems. Covers advanced graphics, easy data input, BASIC enhancements, CP/M cartridge on the '64, POKEs, user defined character sets, joystick/mouse simulation, transferring data between computers, more. A treasure chest.

ISBN-0-916439-03-8 250pp $19.95

GRAPHICS BOOK FOR THE C-64

takes you from the fundamentals of graphic to advanced topics such as computer aided design. Shows you how to program new character sets, move sprites, draw in HIRES and MULTICOLOR, use a lightpen, handle IROs, do 3D graphics, projections, curves and animation. Includes dozens of samples.

ISBN-0-916439-05-4 280pp $19.95

ADVANCED MACHINE LANGUAGE FOR THE C-64

gives you an intensive treatment of the powerful '64 features. Author Lothar Englisch delves into areas such as interrupts, the video controller, the timer, the real time clock, parallel and serial I/O, extending BASIC and tips and tricks from machine language, more.

ISBN-0-916439-06-2 200pp $14.95

IDEAS FOR USE ON YOUR C-64

is for those who wonder what you can do with your '64. It is written for the novice and presents dozens of program listing the many, many uses for your computer. Themes include: auto expenses, electronic calculator, recipe file, stock lists, construction cost estimator, personal health record diet planner, store window advertising, computer poetry, party invitations and more.

ISBN-0-916439-07-0 200pp $12.95

PRINTER BOOK FOR THE C-64

finally simplifies your understanding of the 1525, MPS/801, 1520, 1526 and Epson compatible printers. Packed with examples and utility programs, you'll learn how to make hardcopy of text and graphics, use secondary addresses, plot in 3-D, and much more. Includes commented listing of MPS 801 ROMs.

ISBN-0-916439-08-9 350pp. $19.95

SCIENCE/ENGINEERING ON THE C-64

is an introduction to the world of computers in science. Describes variable types, computational accuracy, various sort alogrithms. Topics include linear and nonlinear regression, CHI-square distribution, Fourier analysis, matrix calculations, more. Programs from chemistry, physics, biology, astronomy and electronics. Includes many program listings.

ISBN-0-916439-09-7 250pp $19.95

CASSETTE BOOK FOR THE C-64

(or Vic 20) contains all the information you need to know about using and programming the Commodore Datasette. Includes many example programs. Also contains a new operating system for fast loading, saving and finding of files.

ISBN-0-916439-04-6 180pp. $12.95

DEALER INQUIRIES ARE INVITED

IN CANADA CONTACT:
The Book Centre, 1140 Beaulac Street
Montreal, Quebec H4R1R8 Phone: (514) 322-4154

AVAILABLE AT COMPUTER STORES, OR WRITE:

Abacus Software
P.O. BOX 7211 GRAND RAPIDS, MI 49510
Exclusive U.S. DATA BECKER Publishers

For postage & handling, add $4.00 (U.S. and Canada), add $6.00 for foreign. Make payment in U.S. dollars by check, money order or charge card. (Michigan Residents add 4% sales tax.)

FOR QUICK SERVICE PHONE (616) 241-5510

Commodore 64 is a reg. T.M. of Commodore Business Machines